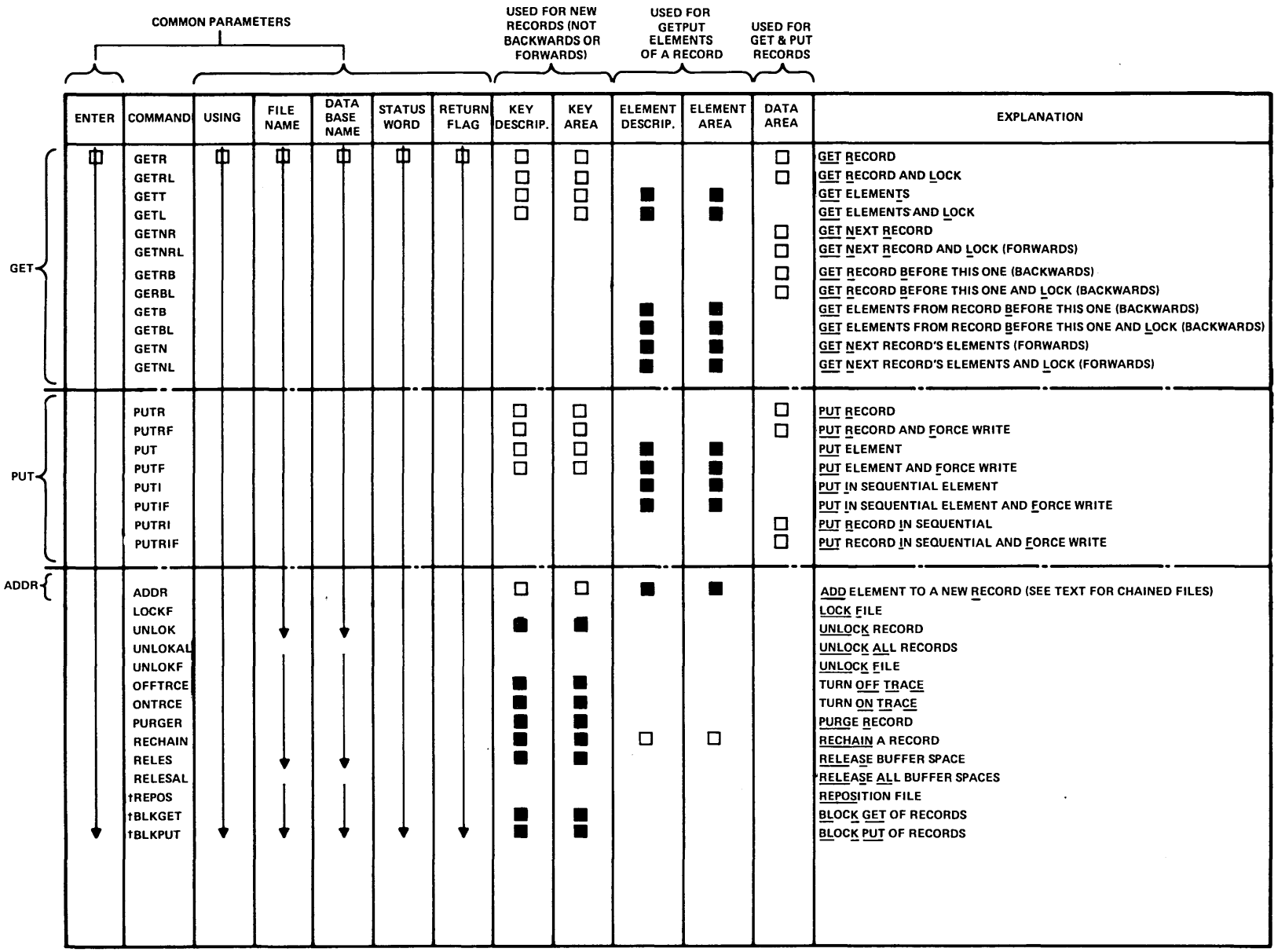
60455350 C

CONTROL DATA CORPORATION

NETWORK PRODUCTS

TRANSACTION FACILITY VERSION 1 DATA MANAGER REFERENCE MANUAL

CDC® OPERATING SYSTEM:
NOS 1



KEY: □ REQUIRED
■ REQUIRED AND MAY BE REPEATED
NO ENTRY IMPLIES NOT ALLOWED
†REQUIRE ADDITIONAL PARAMETERS

60455350

CONTROL DATA CORPORATION

NETWORK PRODUCTS

TRANSACTION FACILITY VERSION 1 DATA MANAGER REFERENCE MANUAL

CDC® OPERATING SYSTEM:
NOS 1

REVISION RECORD

REVISION	DESCRIPTION
A	Manual released for NOS 1.3 at PSR level 472.
(05-12-78)	
B	Revised to update manual to NOS 1.4 at PSR level 501.
(05-18-79)	
C	Revised to update manual to NOS 1.4 at PSR level 531. This edition obsoletes all previous editions.
(12-12-80)	

Publication No.
60455350

REVISION LETTERS I, O, Q AND X ARE NOT USED

Address comments concerning this manual to:

Control Data Corporation
Publications and Graphics Division
4201 North Lexington Avenue
St. Paul, Minnesota 55112

or use Comment Sheet in the back of this manual.

© 1978, 1979, 1980
by Control Data Corporation
All rights reserved
Printed in the United States of America

LIST OF EFFECTIVE PAGES

New features, as well as changes, deletions, and additions to information in this manual, are indicated by bars in the margins or by a dot near the page number if the entire page is affected. A bar by the page number indicates pagination rather than content has changed.

PAGE	REV
Front Cover	-
Inside Front Cover	A
Title Page	-
ii	C
iii/iv	C
v/vi	C
vii	C
viii	C
1-1	C
2-1	C
2-2	C
2-3	C
2-4	C
2-5	C
2-6	C
2-7	C
3-1	C
3-2	C
3-3	C
3-4	C
3-5	C
3-6	C
3-7	C
3-8	C
3-9	C
3-10	C
4-1	C
4-2	C
4-3	C
4-4	C
4-5	C
5-1	C
5-2	C
5-3	C
5-4	C
5-5	C
5-6	C
5-7	C
5-8	C
6-1	C
6-2	C
6-3	C
6-4	C
6-5	C
6-6	C
6-7	C
6-8	C
6-9	C
6-10	C
6-11	C
6-12	C
6-13	C
6-14	C
6-15	C
6-16	C
6-17	C
6-18	C
6-19	C

PAGE	REV
A-1	A
A-2	A
B-1	A
C-1	A
D-1	C
D-2	C
D-3	C
D-4	C
D-5	C
D-6	C
D-7	C
D-8	C
D-9	C
D-10	C
D-11	C
D-12	C
D-13	C
E-1	C
E-2	C
Index-1	C
Index-2	C
Comment Sheet	C
Back Cover	-

PAGE	REV

PAGE	REV

PAGE	REV

PREFACE

The CONTROL DATA Transaction Facility (TAF) is a network host product. This manual describes the data manager available under TAF. It does not explain file backup and recovery design; only the features used to implement such design are described.

AUDIENCE DEFINITION

The reader is required to have only a limited knowledge of the TAF executive. The executive is described in the TAF Reference Manual. A new user should read the TAF User's Guide or the TAF Reference Manual before reading this manual. The reader should be familiar with basic NOS concepts such as running a compiler job. A knowledge of FORTRAN, COBOL, or COMPASS is also required.

TERMINOLOGY

Within this manual, references to the reference manual imply the TAF Reference Manual. References to FORTRAN imply FORTRAN Extended Version 4; references to COBOL imply both COBOL Versions 4 and 5 unless specifically differentiated.

RELATED PUBLICATIONS

The NOS Manual Abstracts is a pocket-sized manual containing brief descriptions of the contents and the intended audience of all NOS and NOS product manuals. The abstracts can be useful in determining which manuals are of greatest interest to a particular user.

Control Data also publishes a Software Publications Release History of all software manuals and revision packets it has issued. This historical report of the manual lists the revision level of a particular manual that corresponds to the level of software installed at the site.

Control Data Publication	Publication Number
Network Products Transactions Facility Version 1 Reference Manual	60455340
Network Products Transaction Facility Version 1 User's Guide	60455360
COBOL Version 4 Reference Manual	60496800
COBOL Version 5 Reference Manual	60497100
FORTRAN Extended Version 4 Reference Manual	60497800
COMPASS Version 3 Reference Manual	60492600
NOS Version 1 Installation Handbook	60435700
NOS Manual Abstracts	84000420
Software Publications Release History	60481000

DISCLAIMER

This product is intended for use only as described in this document. Control Data cannot be responsible for the proper functioning of undescribed features or undefined parameters.

CONTENTS

1. INTRODUCTION TO THE TAF DATA MANAGER 1-1

Device Independence 1-1
Unique File Names 1-1
Security 1-1

2. DATA MANAGER FILE STRUCTURE 2-1

Chaining and Secondary Keys 2-1
- Simple Chaining 2-1
- Complex Chaining 2-2
- Secondary Keys 2-3

Record Accessing Techniques 2-3
File Types 2-4
- Sequential File 2-4
- Random File 2-5
- Direct-Key-Access (Preallocated) File 2-5
- Direct-Key-Access Indexed File 2-5
- Hashed-Access Indexed File 2-7

3. DATA BASE DEFINITION (DATADEF) 3-1

DATADEF Statements 3-1
- DATADEF Control Statement 3-1
- Output Formatting Statements 3-2
  - COMMENT Statement 3-2
  - EJECT Statement 3-2
  - PREFIX Statement 3-2
  - SPACE Statement 3-2
  - TITLE Statement 3-2

DATAMAP Utility 3-2
- Data Base (DATAMAP) Utility 3-2
- DATAMAP Control Statement 3-4

Input to DATADEF 3-4
- Format Specifications for DATADEF Input 3-4
- Notation and Symbols for DATADEF Descriptions 3-4
- Input Sections 3-5
  - Identification Section 3-5
  - Record-Description Section 3-5
  - File-Description Section 3-7
  - File-Relationship Section 3-9

4. DATA BASE INFORMATION (DBFORM) 4-1

General Functions of DBFORM 4-1
- Create Data Base 4-1
- Purge Data Base 4-1
- Reformat Data Base 4-1
- Trial Reformat of Data Base 4-2

DBFORM Control Statement 4-2
- Description of Data Base Parameter (DB=xx) 4-2
- Description of Run Options (OP=yy) 4-2
  - C - Create New Data Base 4-2
  - P - Purge Data Base 4-3
  - R - Reformat Data Base 4-3
  - TR - Trial Reformat of Data Base 4-3
- Description of Remaining Parameters (p1,p2,...pn) 4-4
  - D - Dump File for Reformat Option 4-4
  - L - Load File for Reformat Option 4-4
  - LO 4-4
  - N - New EDT File 4-4
  - R - Recovery Parameters 4-4
  - I - Input Directive File 4-4

Input Directives 4-4
DBFORM Error Messages 4-5
- Element Processor Error Messages 4-5

Informative Messages 4-5

5. DATA MANAGER COMPONENTS AND FILE BACKUP 5-1

Data Manager Components 5-1
- Control Program 5-1
- File Access Processor 5-2
- Function Processors 5-2
  - Basic Function Processors 5-2
  - Basic Function Processor Options 5-2
  - Additional Function Processors 5-3
- Element Processor 5-4

File Backup 5-4
- Dual-Record 5-5
- Trace Files 5-6
- Pool Files 5-6

6. DATA MANAGER COMMANDS 6-1

Command Parameter Definitions 6-1
Special Considerations for COBOL 4 and 5 Data Types 6-4
Commands 6-4
- ADDR 6-5
- BLKGET 6-5
- BLKPUT 6-5
- CLEAR 6-6
- DMSTAT 6-6
- GETB 6-6
- GETBL 6-6
- GETN 6-7
- GETNL 6-7
- GETR 6-7
- GETRL 6-7
- GETNR 6-7
- GETNRL 6-8
- GETRB 6-8
- GETRBL 6-8
- GETT 6-8
- GETL 6-8
- LOCKF 6-9
- OFFTRCE 6-9
- ONTRCE 6-9
- PURGER 6-9
- PUT 6-9
- PUTF 6-10
- PUTI 6-10
- PUTIF 6-10
- PUTR 6-10

PUTRF 6-11
PUTRI 6-11
PUTRIF 6-11
READEDT 6-11
REPOS 6-11
RECHAIN 6-12
RELES 6-12
RELESAL 6-12
STATCHK 6-13
UNLOK 6-13
UNLOKAL 6-13
UNLOKF 6-13
Packed Format Calling Sequence 6-13
Error Processing 6-15
Batch Access to the Data Manager 6-17
Trace Option 6-17
Summary of Commands 6-17

APPENDIXES

A. BST STATUS A-1
B. ELEMENT CONVERSION PROCESSING B-1
C. KEY CONVERSION ACCESS METHODS C-1
D. MESSAGES D-1
E. SAMPLE DECK STRUCTURES E-1

INDEX

FIGURES

2-1 Sequential Simple Chained File 2-2
2-2 Indexed Simple Chained File 2-2
2-3 Complex Chained File 2-2
2-4 Chainer File 2-2
2-5 Secondary Keys 2-3
2-6 Summary of File Types 2-4
2-7 Sequential File 2-5
2-8 Example of a Random GET 2-5
2-9 Example of a GET on a Prelocated File 2-6
2-10 Direct-Key-Access Indexed File 2-6
2-11 Hashed-Access Indexed File 2-7
3-1 General Format of Record Description Section 3-5
3-2 File Types Supported by the Transaction System 3-9
5-1 Record Status Entry Format 5-3
5-2 Simple of Key Element Descriptor Format 5-4
5-3 Group Element Descriptor Format 5-5
5-4 Element Conversion Legality 5-5
5-5 Trace File Structure 5-7
5-6 Trace File Header Format 5-7
5-7 Error Recovery Pool File Structure 5-8
5-8 Error Recovery Pool File Record Header Format 5-8
6-1 Normal Call Versus a Packed Format Call 6-16
6-2 Trace Request Format 6-18

TABLES

6-1 Data Types 6-4
6-2 Preventive Errors - Message Field 6-14
6-3 Nonpreventive Errors - Non-fatal Error Field 6-15
6-4 CTI and SCT Requests 6-19
6-5 Operating Differences 6-19

The TAF data manager can be used in either batch mode or transaction mode. It is designed to be used with the transaction subsystem in an interactive manner; however, batch use is allowed for initializing and testing a data base.

Each application consists of one or more data bases. The user may choose to use as many data bases as he wishes. An application may access any data base subject to the normal security provisions of the system and subject to optional constraints within TAF.

A data base is a collection of files. The user may have as many files associated with a data base as he wishes. The data manager allows five different file types so that the user can choose a file structure most suited to the data being processed. The files are all direct-access, mass-storage files.

Each file consists of records. Within a file, all records must have the same structure. There is no limit to the number of records per file. Records in a file may be linked together by chains, and data elements in one record can be used to access records in another file.

DEVICE INDEPENDENCE

The organization of data within the data manager is independent of the mass-storage medium. Device independence has the following advantages.

- Data files may be contained on any mass-storage device.
- The user is not tied to any physical equipment configuration after assembling or compiling a program. In fact, new equipment can be added to an installation without reprogramming or recompiling any application programs.
- Within the constraints imposed by the operating system, the user may experiment with different combinations and types of I/O devices to achieve optimum performance.

UNIQUE FILE NAMES

To avoid duplicate file names, each user should be assigned a unique data base name. The file names associated with the data manager are a combination of the data base name and the user's file name. This is done when the files are defined by DBFORM or the user. The two leftmost characters of the system file name are the data base name; the next four characters are the user's file name. Files must not require passwords for accessing.

The TAF configuration file (TCF), an indirect-access permanent file containing the names of all data bases supported, is maintained under the transaction subsystem user number. This file is described in the reference manual. The file is not needed for batch use of the data manager.

SECURITY

Each data base may be assigned to a different user number. This capability increases security by preventing one user from accessing files belonging to another user's data base.

Illegal data base accessing is prevented in the on-line mode through the use of the data base name field in the terminal network file. (Refer to the Installation Handbook for a description of this file. Refer to preface for publication number.) This field contains the name of the data base assigned to the particular terminal name. It is retrieved from the validation file when the transaction subsystem is initialized and checked by the data manager on each data base request made by a terminal. An error indicating illegal access is returned any time a terminal attempts to access a data base for which it is not validated. This limits each terminal defined in the validation file to one and only one data base. However, provision is made to allow certain terminals to have access to all data bases by entering the characters SY in the validation file for the data base name. The user cannot inadvertently use the SY data base name.

In addition, each terminal name has a read-security and an update-security associated with it. To read an element in the data base, the terminal must have a read-security value equal to or greater than the read-seucrity value defined with the element. The same is also true for updating an element.

For jobs using the batch data manger, the operating system permanent file security checking is available. This means that if a user knows which user number the data base is under and attaches those data base files needed, he is validated to use the data base.

There are two general methods of accessing a file: sequential and random. In sequential access of a file, the user can access only the previous record or the following record. He cannot access any record other than in relationship to the previous record retrieval or in relationship to the beginning or end of the file. An example of a sequential file is a tape file. All file types supported by the data manager can be accessed sequentially.

In random access of a file, the user can access any record in the file by supplying a key (that is, an alphanumeric value) that identifies the record; the data manager then locates that record for processing. Random files differ from each other in two major ways: first, in the structure and range of values allowed for the key, and second (partially determined by the first difference) in whether the file is of fixed length (preallocated) or of variable length (dynamically allocated). Random files may be accessed sequentially or randomly. The data manager extends the concept of sequential access of a random file by allowing simple and complex chaining.

To access a file sequentially, the records must be linked by a chain. A chain is one pointer to the previous record and a second pointer to the following record. The pointers are assigned by the data manager and added to the record. The chain is needed because records are not necessarily physically stored in their correct sequence. Chains are not described in the definition of a record's structure. All data manager files have a simple chain.

Complex chaining allows a random file to be subdivided into two or more chains. This means that a user can sequentially process a file using only those records assigned to the chain. For example, a file containing an inventory of merchandise in warehouse A would be in one chain; all the inventory entries for merchandise in warehouse B would be in a second chain. The user then could sequentially access all the inventory entries (records) belonging to a specific warehouse. When the same file is accessed randomly, the complex chain has no meaning. A complex chain must be specifically defined when the record and file structures are described.

Files may also use a data element from one record as the key to retrieve a record from another file. This technique, called secondary keys, is allowed for all except sequential files.

The data manager supports five file types as follows:

- Sequential

  A sequential access file that is dynamically allocated. Complex chains are not allowed.

- Random

  A random or sequential access file that is dynamically allocated. The keys are generated by the data manager. Complex chains are allowed.

- Direct-key-access

  A random or sequential access file that is preallocated. The keys are generated by the user. Complex chains are allowed.

- Direct-key-access-indexed

  A random or sequential access file that is dynamically allocated. The keys are generated by the user and are kept in a separate index file that is preallocated. Complex chains are allowed.

- Hashed-access-indexed

  A random or sequential access file that is dynamically allocated. The keys are generated by the user and are kept in a separate index file that is dynamically allocated. Complex chains are allowed.

The remainder of this section describes chaining, secondary keys, and the five file types in greater detail.

CHAINING AND SECONDARY KEYS

SIMPLE CHAINING

To provide chaining, the data manager reserves the first word (60 bits) of each record. This word is split in half, with each half containing a relative physical record unit (PRU) number (the pointer). The left half of the word contains the pointer to the next record forward in the sequence; the right half contains the pointer to the prior record in the sequence. If the right half of the word is zero, the record is the start of the chain. When the left pointer is zero, the record is the end of the chain. Thus, the first record points forward to the second record and has a zero backward pointer. The second record points forward to the third record and backward to the first record. This linkage continues to the last record. The last record has a zero forward pointer, but its backward pointer links to the last record minus one.

The first word (word 0) is always reserved by DATADEF, so the user need not reserve it. The user need only remember that it exists when assigning the overall length of a record. The user may not access the chain word.

Figure 2-1 is an example of a four-record file that has a simple chain.

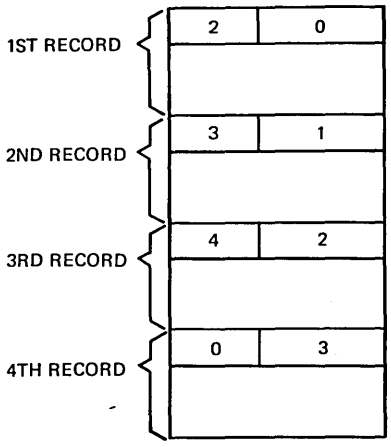


Figure 2-1. Sequential Simple-Chained File

The records in a simple-chained file are not necessarily sequentially contiguous if the file is indexed. The order in which the records occur in the file is the same as the order in which they were added by the user (through the use of the ADDR request). A simple-chained preallocated indexed file might appear as figure 2-2.

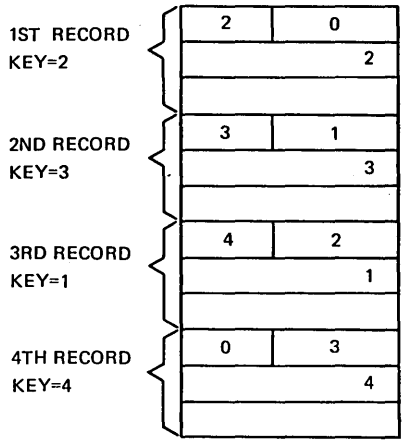


Figure 2-2. Indexed Simple-Chained File

COMPLEX CHAINING

Complex chaining allows the user to divide a data file into more than one chain. A record in a data file is contained in only one chain. Each data record in a complex-chained file contains an element, the chaining element. The value of the chaining element is used to determine to which chain the record belongs. As with simple chaining, the first word of a record contains pointers to the previous and next records in the chain. The backward pointer of the first record of the chain is zero. The forward pointer of the last record of the chain is zero. A separate file, the chainer file, controls the chains by maintaining pointers to the first and last records of each chain. A one-to-one correspondence exists between the records in the chainer file and the chains in the data file. The key value of a given record in the chainer file is the same value as the chaining elements in the records of the corresponding chain of the data file. A group element, the control element, in a given chainer file record contains the values of the keys of the first and last records in the corresponding chain.

For example, a college might keep all of its student records, both graduate and undergraduate, on one file named STUDENT. The file is divided into two chains: one containing the records of the graduate students and another containing the records of the undergraduate students. The chaining element of this file is STUDENT-TYPE with possible values of GRADUATE and UNDERGRADUATE. The primary key of file STUDENT is STUDENT-NAME. Figure 2-3 shows the possible contents of the file STUDENT.

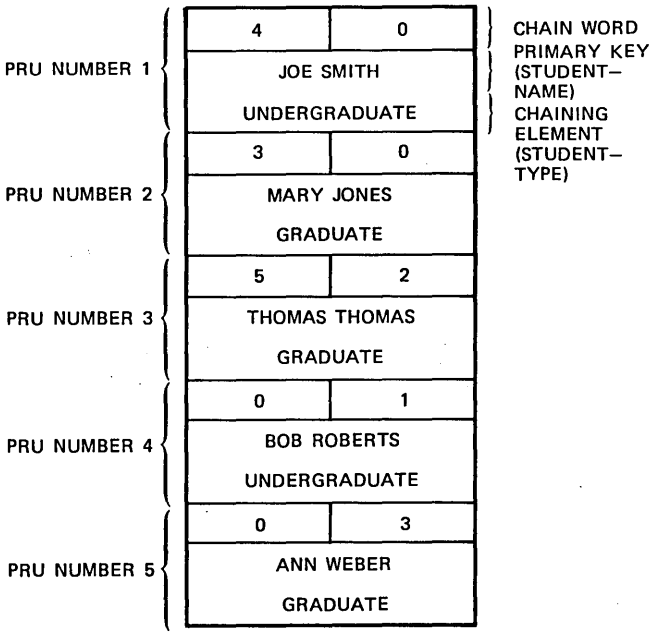


Figure 2-3. Complex-Chained File (STUDENT)

The file STUDENT-LEVEL is the chainer file for the data file STUDENT. The primary key of file STUDENT-LEVEL is LEVEL, which has two possible values: GRADUATE and UNDERGRADUATE. The control element FIRST-LAST contains pointers to the first and last records of the chain corresponding to LEVEL. Figure 2-4 shows the chainer file STUDENT-LEVEL.

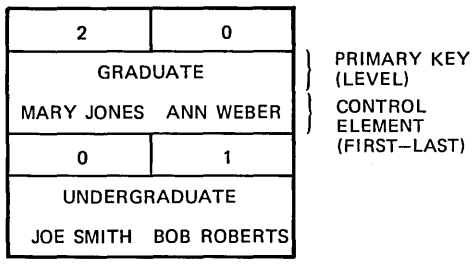


Figure 2-4. Chainer File (STUDENT-LEVEL)

One chainer file can control more than one chained data file. To accomplish this task, the user must define more than one control element in the chainer file record (one control element for each chained data file).

The user must initialize the chainer file by adding empty records with the desired primary key values. The data manager automatically updates the values of the control elements in the chainer file when records are added to, deleted from, or rechained in the chained data file.

The user can access all the records in a given chain using the REPOS command (section 6). The user can access each individual record in a given chain using a series of GETN commands (section 6).

SECONDARY KEYS

Secondary keys provide a means of finding records in one file through values of an element in another file. An element defined as the secondary key of one file contains the same data as an element defined as the primary key of another file. The second file also contains an element that contains the same data as the primary key of the first file.

For example, assume two files, one called SAVE and the other called LOAN, contain somewhat related data. File SAVE contains an element named ANO which is the savings account number, and an element called NAM which is the name of the person owning the account. ANO is the primary key of file SAVE. File LOAN contains a primary key element called NM1 which is the name of the borrower and an element called AN1 which contains the savings account number. Elements ANO and AN1 have the same description, as do elements NAM and NM1. Figure 2-5 illustrates this file structure.

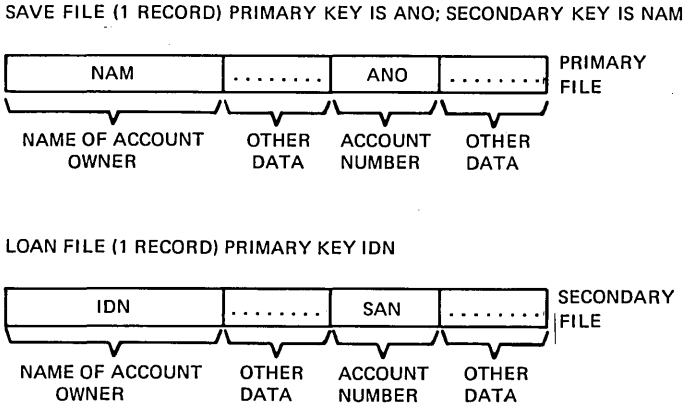


Figure 2-5. Secondary Keys

To access a record in file SAVE when the key ANO (account number) is not known but the name is known, the user can issue a GET type request for file SAVE specifying NAM as the key descriptor. The data manager recognizes element NAM as a secondary key, finds the record in file LOAN where the primary key NM1 matches the value specified for NAM, picks up the value for AN1 from the record, and accesses the record from file SAVE that has a matching value for ANO.

The data manager uses the secondary key method to read or modify records, but cannot use it to create or maintain records on the secondary file. In the example, if SAVE is the main file and LOAN is the secondary file, all maintenance of file LOAN must be done by the user.

For example, when a record is added to the SAVE file, it is the user's responsibility to update the LOAN file to show that new record. No automatic linking is done by the data manager.

Example:

ENTER GETR USING SAVE data base name, status word, return flag, NAM, key area, data area.

- The secodary key NAM is located and used by the data manager as a primary key to get a record from LOAN.
- The data manager locates secondary key AN1 and uses it as the primary key to get the record from SAVE.

RECORD ACCESSING TECHNIQUES

A user has two general methods of accessing records in the various files: sequential and random. The sequential method repositions to the start or end of a file and uses one of the get next (for example, GETN) requests to search forward, or one of the get back (for example, GETB) requests to search backward. These requests read through the chain until the next active record or the end of the chain is found. The reposition command can be used to reposition to the beginning or end of a file. If the file is complex-chained, it can also be repositioned to the beginning or end of a specific chain. For specific information on the get and reposition functions, refer to section 6.

The random method of accessing records uses a key value that is associated with a specific record in the file. This method is not applicable to sequential files since they have no key specified. All other file types have keys specified. Keys are specified during data base definition. To use the key method of record access, the user must pass the key to the data manager as an explicit or implicit logical record number.

The only exception is an add record operation (ADDR) on a random file type where the relative PRU number is returned to the user, in contrast to the same operation on a direct-key-access file where the key is passed by the user to the data manager. If the user has defined a key conversion access method for his file during data definition (refer to Access-Method, Bias-Value in section 3), he specifies an implicit logical record number which is converted to an explicit logical record number by the access method. If no access method was defined, the key value is an explicit logical record number. This key value, or the value returned by a user-defined access method, is converted to a relative PRU number according to the file type.

A key conversion access method is a user-provided subroutine; it converts a key value into a format that can be used by the data manager. For example, the last two digits of an individual's name could be used as an octal value to access a record. The access method would strip all but the last two characters and shift them to the lower 6 bits of the 60-bit word. The user would send the individual's name as a key to the data manager which would invoke the access method and use the resulting 60-bit number to get or put the record. An access method can be used for all file types except sequential files. The access method to be used for a particular file is specified in File-Description Section, section 3.

When using a key conversion subroutine, it is important that the converted key value be in the proper format for the type of file being accessed. The following are the formats for the various types of files.

File Type	Key Format
Random	Relative PRU number
Direct-key-access (preallocated)	Logical record number
Direct-key-access (indexed)	Logical record number
Hashed-access indexed	Relative PRU number of index record

All converted key values must be unsigned integers.

Appendix C describes the procedure to insert a key conversion subroutine into the system and shows a sample access method.

FILE TYPES

Figure 2-6 summarizes the files and gives the circumstances under which each one would be used. Note that all records within a file must have the same structure. Also, with the exception of the sequential file, a key is used to access each record within the file. The definition of record and file structure is done by the DATADEF utility which is decribed in section 3.

SEQUENTIAL FILE

No key is used with this type of file. A simple chain is maintained by the data manager. The file can be accessed only by positioning to the beginning or end of this chain, by the use of REPOS, and by using either the next or back option to obtain the data desired. Once the file has been accessed, any sequence of back or next type functions is allowed without further positioning as the data manager maintains the user's current position in the file. When the file has been positioned to the beginning, a next type request obtains the first record. When a file has been positioned to the end, a back type request retrieves the last record in the file.

The structure of a sequential file of four records is shown in figure 2-7.

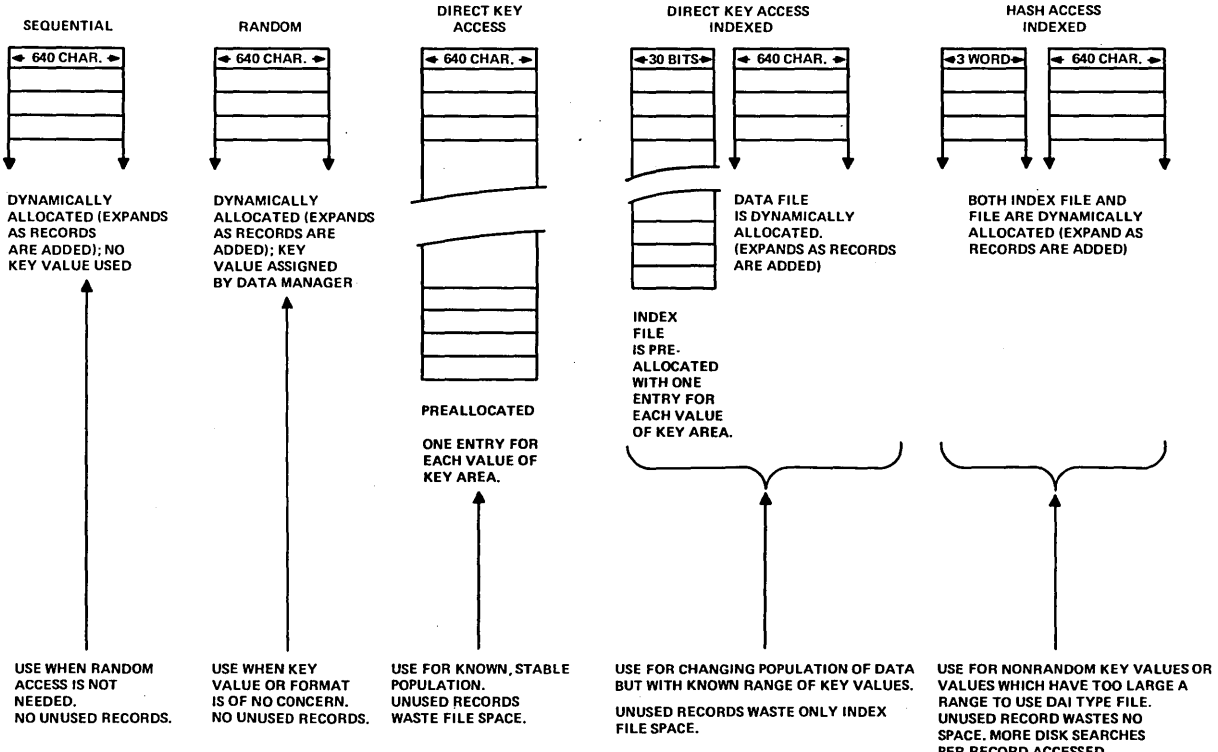

SEQUENTIAL
640 CHAR.
DYNAMICALLY ALLOCATED (EXPANDS AS RECORDS ARE ADDED); NO KEY VALUE USED
USE WHEN RANDOM ACCESS IS NOT NEEDED. NO UNUSED RECORDS.
RANDOM
640 CHAR.
DYNAMICALLY ALLOCATED (EXPANDS AS RECORDS ARE ADDED); KEY VALUE ASSIGNED BY DATA MANAGER
USE WHEN KEY VALUE OR FORMAT IS OF NO CONCERN. NO UNUSED RECORDS.
DIRECT KEY ACCESS
640 CHAR.
PREALLOCATED
ONE ENTRY FOR EACH VALUE OF KEY AREA.
USE FOR KNOWN, STABLE POPULATION. UNUSED RECORDS WASTE FILE SPACE.
DIRECT KEY ACCESS INDEXED
30 BITS
640 CHAR.
DATA FILE IS DYNAMICALLY ALLOCATED. (EXPANDS AS RECORDS ARE ADDED)
INDEX FILE IS PRE-ALLOCATED WITH ONE ENTRY FOR EACH VALUE OF KEY AREA.
USE FOR CHANGING POPULATION OF DATA BUT WITH KNOWN RANGE OF KEY VALUES.
UNUSED RECORDS WASTE ONLY INDEX FILE SPACE.
HASH ACCESS INDEXED
3 WORD
640 CHAR.
BOTH INDEX FILE AND FILE ARE DYNAMICALLY ALLOCATED (EXPAND AS RECORDS ARE ADDED)
USE FOR NONRANDOM KEY VALUES OR VALUES WHICH HAVE TOO LARGE A RANGE TO USE DAI TYPE FILE. UNUSED RECORD WASTES NO SPACE. MORE DISK SEARCHES PER RECORD ACCESSED.

Figure 2-6. Summary of File Types

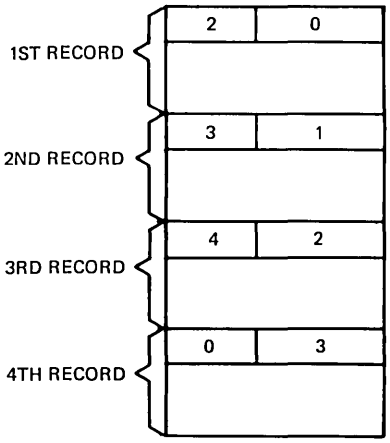


Figure 2-7. Sequential File

RANDOM FILE

Random files are the only type of files that have a key where the user does not specify the key when the record is initially opened with an add record (ADDR) request. The record if added at the end-of-information (EOI). The relative PRU number is passed back to the user in the key area specified in the request. This relative PRU number is used as a key for all subsequent requests. Figure 2-8 shows an example of a random GET. There is no logical limit to the number of records in this type of file. However, the random file key format limits the relative PRU number to a 30-bit integer, right- justified in the field.

DIRECT-KEY-ACCESS (PREALLOCATED) FILE

The key value (or the value returned from a key conversion access method) is a logical record number. The data manager converts the logical record number to a relative PRU number by multiplying the number of PRUs in one record by the logical record number minus 1, and then adding 1 to this product. Figure 2-9 is an example of a GET on a preallocated file. The user is restricted to access only those records which were preallocated according to the repeated clause in DATADEF at the time the data base was defined (refer to REPEATED Clause in section 3).

DIRECT-KEY-ACCESS INDEXED FILE

The key value (or the value returned from an access method) is a logical entry number in the index file. The data manager uses the key value to access the proper index record and entry within this record to find the relative PRU number for the desired data record. A direct-key-access index record is 64 words long with two 30-bit entries in each word. Thus, there are 128 entries per index record that contain relative PRU numbers of records in the data file. The key given by the user is divided by 128, and 1 is added to the quotient. The integer portion of the quotient points to the relative PRU number in the index file. The numerator of the remainder points to the entry in the index record which contains the relative PRU number of the desired data record. The logical limit to the number of records that may be handled is 128 times the number of index records specified in the ENTRIES clause at data base definition time. Figure 2-10 illustrates access on this file type.

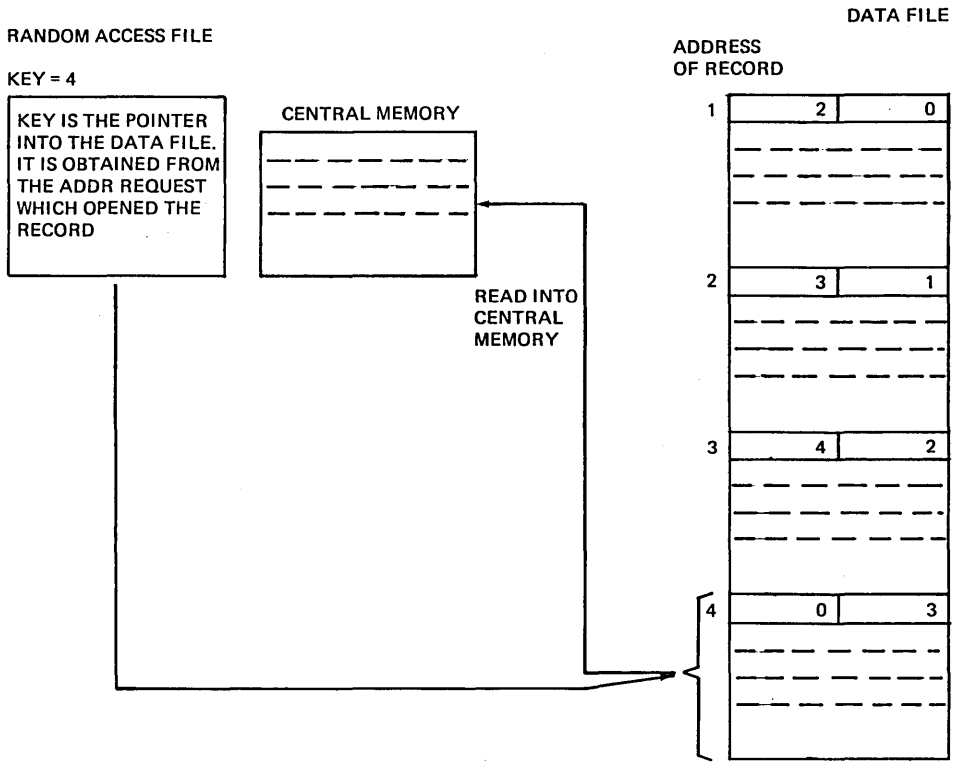


Figure 2-8. Example of a Random GET

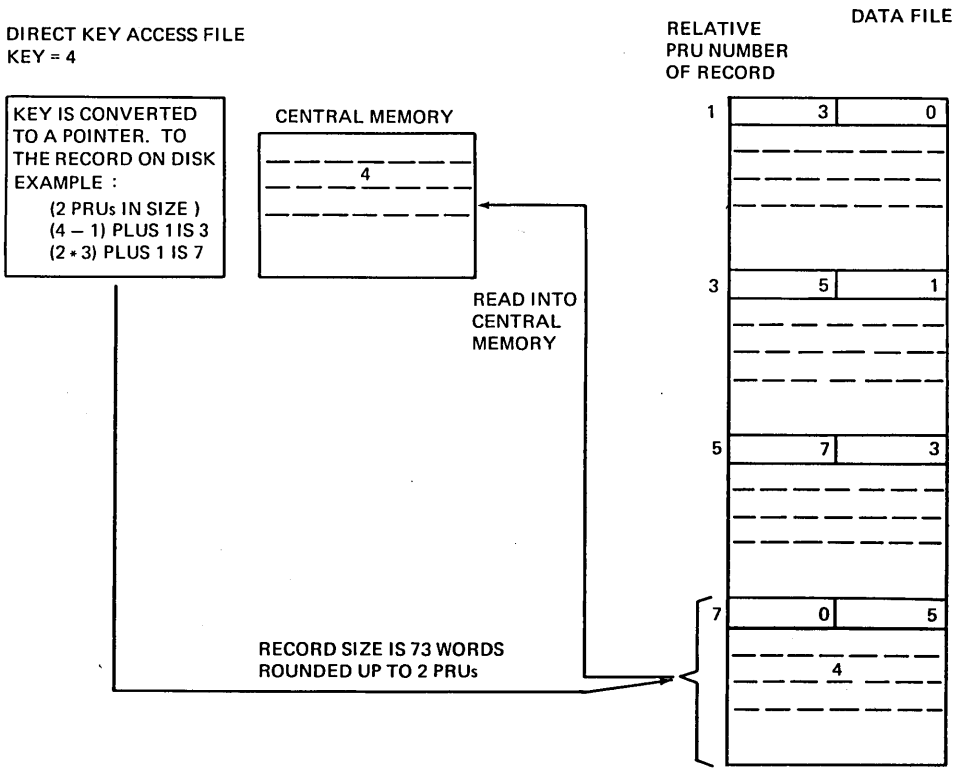


Figure 2-9. Example of a GET on a Preallocated File

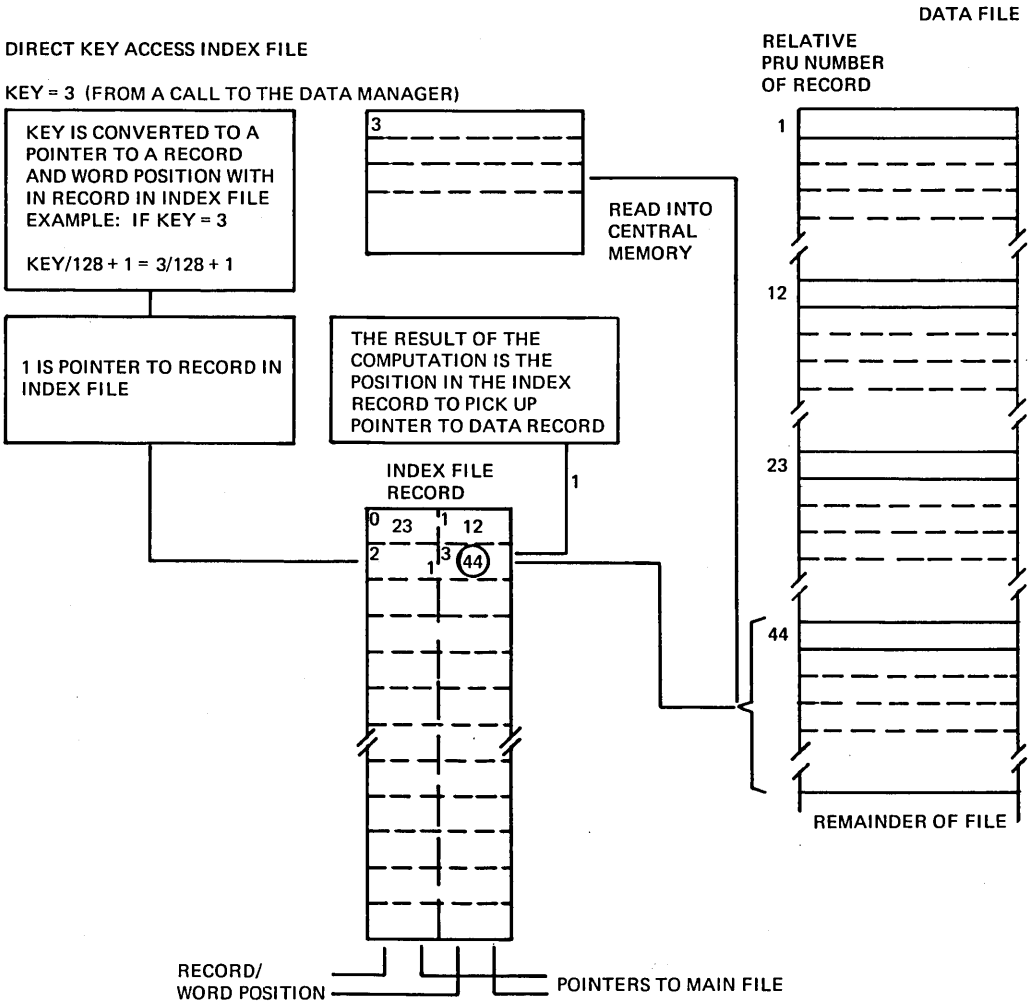

DIRECT KEY ACCESS INDEX FILE
KEY = 3 (FROM A CALL TO THE DATA MANAGER)
KEY IS CONVERTED TO A POINTER TO A RECORD AND WORD POSITION WITH IN RECORD IN INDEX FILE EXAMPLE: IF KEY = 3
KEY/128 + 1 = 3/128 + 1
1 IS POINTER TO RECORD IN INDEX FILE
THE RESULT OF THE COMPUTATION IS THE POSITION IN THE INDEX RECORD TO PICK UP POINTER TO DATA RECORD
INDEX FILE RECORD
3
READ INTO CENTRAL MEMORY
1
0 23 1 12
2 3 (44)
1
RECORD/ WORD POSITION
POINTERS TO MAIN FILE
DATA FILE
RELATIVE PRU NUMBER OF RECORD
1
12
23
44
REMAINDER OF FILE

Figure 2-10. Direct-Key-Access Indexed File

HASHED-ACCESS INDEXED FILE

The key to be hashed (such as a social security number) is sent to the data manager. The access method then hashes that key to a relative PRU number of a record in the index file. Each index record contains a maximum of 21 three-word entries, each of which is a pointer to a data record.

An entry is three words long. The first and second words contain the original key value (even if a key conversion access method is used) and the third word contains the relative PRU number of the associated data record. Each index record is 64 words long. The last word in the record is a relative PRU number of an overflow record in the index file for this index pointer. If more than 21 entries are made for this index pointer, an overflow record is appended at the end of the index file.

When the index record is obtained, a search is performed on the original key value to find the proper entry for the data record. If necessary, overflow records are searched.

An index pointer can have as many overflow records as needed. The number of base index records in the file is specified by the ENTRIES clause at data base definition time.

Hashed-access indexing is normally used in situations where the access† method cannot convert to a unique value (for example, using the first two letters of a last name). Ideally, the conversion method maximizes the use of space in index records without causing the creation of overflow records.

Figure 2-11 illustrates access on this file type.

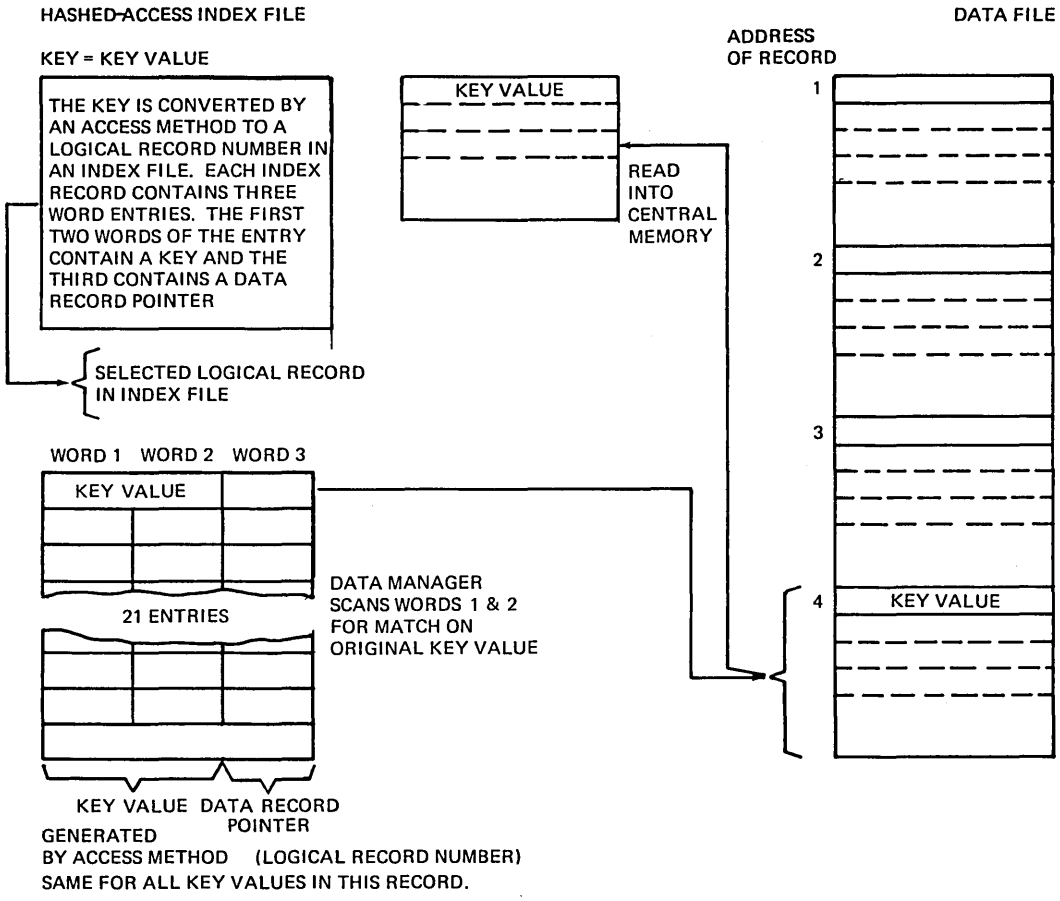

HASHED-ACCESS INDEX FILE
KEY = KEY VALUE
THE KEY IS CONVERTED BY AN ACCESS METHOD TO A LOGICAL RECORD NUMBER IN AN INDEX FILE. EACH INDEX RECORD CONTAINS THREE WORD ENTRIES. THE FIRST TWO WORDS OF THE ENTRY CONTAIN A KEY AND THE THIRD CONTAINS A DATA RECORD POINTER
SELECTED LOGICAL RECORD IN INDEX FILE
WORD 1 WORD 2 WORD 3
KEY VALUE
21 ENTRIES
KEY VALUE DATA RECORD POINTER
GENERATED BY ACCESS METHOD (LOGICAL RECORD NUMBER) SAME FOR ALL KEY VALUES IN THIS RECORD.
KEY VALUE
READ INTO CENTRAL MEMORY
DATA MANAGER SCANS WORDS 1 & 2 FOR MATCH ON ORIGINAL KEY VALUE
DATA FILE
ADDRESS OF RECORD
1
2
3
4
KEY VALUE

Figure 2-11. Hashed-Access Indexed File

†While an access method is not mandatory, most applications will require one.

The data manager maintains multiple data bases. However, a terminal can be limited to the data bases it accesss when the network file is defined.

Each data base under data manager control is described by an element descriptor table (EDT) file. Before being created, this file must be defined (using the DEFINE control statement) as a direct-access permanent file on the application program user index. The EDT file name is identical with the two-character data base name. A list of data base names currently supported by the transaction system is maintained on a file named TCF, which is stored on the transaction executive user index.†

The DATADEF utility defines a data base. The formation of the data base is done by the DBFORM utility (refer to section 4). Output from DATADEF consists of a listing and the binary EDT file.

This section discusses the following topics.

- DATADEF statements
  - Control statement
  - Output formatting statements
- DATAMAP
  - DATAMAP utility
  - DATAMAP control statement
- Input to DATADEF
  - Format specifications
  - Notation and symbols for DATADEF descriptions
  - Input sections
- DATADEF error messages

DATADEF STATEMENTS

The DATADEF utility is invoked by the following control statement.

DATADEF CONTROL STATEMENT

The DATADEF(p_1,p_2,p_3,...p_n)

p_i can be:

I Source input file

I	Source on COMPILE
I=fn	Source on user-assigned file name fn
absent	Source on INPUT

L List file

L	List on LIST
L=fn	List on user-assigned file name
L=0	No list
absent	List on OUTPUT

B Binary output file (EDT file)

B	Binary output on LGO
B=fn	Binary output on user-assigned file name
B=0	No binary output
absent	Binary output on LGO

P Compiler control statement prefix††

P	Prefix =
P=c	Prefix c
P=0	No compiler control statements
absent	Prefix (

O Short list file (errors and statistics only)

O	Short list on OUTPUT
O=fn	Short list on user-assigned file name
O-0	No short list
absent	No short list

NA Compilation error abort control

NA	Do not abort
absent	Abort job on error

LO List options

LO	List compiler control statements (same as LO=C)
LO=C	List compiler control statements
absent	No change to assumed option settings

M Generate data base map†††

M	List map on file specified for L
M=fn	List on map user-assigned file name
M=0	No map
absent	No map

†Refer to reference manual.
††Refer to Output Formatting Statements in this section.
†††A data base map can also be listed by use of the DATAMAP control statements described later in this section.

OUTPUT FORMATTING STATEMENTS

The P parameter on the DATADEF statement permits the user to control the listing of the source file, or to control certain parts of the binary output file (EDT). The use of this option requires one of the following source language statements available under DATADEF.

COMMENT cc....c

EJECT cc....c

PREFIX c

SPACE,n,cc....c

TITLE cc....c

The P parameter specifies:

- Whether there are any compiler control statements in the input deck (P=0 means there are none).
- If there are compiler control statements, which prefix character will be used in the control statements. For instance:

  CEJECT cc....c

  appearing as a control statement indicates that P=C was selected on the DATADEF card. Explanations of each control statement follow.

The compiler control statements are inserted in the input deck for the data base.

COMMENT Statement

This statement places a comment in the prefix table of the binary file. The text for the comment begins at the first nonblank character after COMMENT.

Comments are placed in the prefix table in a circular fashion. If the prefix table becomes full, the next comment is placed starting at word 3. These comments are useful to utility programs such as DATAMAP, CATALOG, and LIBEDIT to provide identification of the record.

EJECT Statement

This statement ejects the page and sets the subtitle field with the text cc....c following the EJECT statement.

PREFIX Statement

This statment overrides the P parameter on the DATADEF statements by setting the compiler control prefix to c. The c on the statement must be the second character after PREFIX.

SPACE Statement

This statement causes the printer to space n lines on the listing. It also causes the text specified to be inserted in the subtitle field. The spacing does not extend past the top of the next page.

TITLE Statement

This statement specifies the title of the listing. The text of the title begins at the first nonblank character after TITLE.

DATAMAP UTILITY

DATA BASE (DATAMAP) UTILITY

If M or M=filename is specified on the DATADEF control statement, the data base being defined is listed (mapped) for the user at the completion of data base definition. The data base map contains detailed file descriptions of all files in the data base. This data base map can also be listed by use of the DATAMAP control statement described later in this section.

Each page of the data map listing contains a two-line header. The first line contains the data base name, map date, map time, and page number. The second line contains descriptive information taken from the prefix table of the element description table record, the last listed file name, and the last listed element name. Prefix information may be inserted by using the COMMENT control statement in the DATADEF input deck.

The first page of data map output contains an alphabetical list of files in the data base. Next to each file name is a number indicating the number of the page in the data map listing that contains a description of that particular file. Following is a sample of output as it might appear on the first page of a data map listing.

```
DATA BASE ZA                DATAMAP - VER1.

        FILES IN DATA BASE.

        DZACUST  *  ZABRAN  2  ZABRCH  2
        ZACHAN  2  ZACUST  3

        ZAERPF  *  ZAINDX  3  ZAKAYF  2
        ZARNDM  2  ZATFIL  *
```

Implicit files, if there are any, are also listed on this page. Implicit files are files that are not specifically named in the element descriptor table, but are required in the data base. These include such files as:

- Secondary copies of dual-recorded files
- Trace file if at least one file is to be traced
- Error recovery pool file

An implicit file is identified in the list by the presence of an asterisk in lieu of a page number. Descriptions of implicit files are not included in the data map listing, and the page number is therefore not applicable.

The second page and all continuing pages of the data map contain descriptions of the explicit files that were listed on the first page of the data map. Each file is introduced by one or two lines describing the file characteristics. In all cases, the file name is the leftmost field on the first of these lines, and the file type follows. Some file types, such as the following, are abbreviated.

DIR. ACC. Direct-key-access

D. A. I. Direct-key-access indexed

H. A. I. Hashed-access indexed

Following the file name and type are various other attributes that vary by file type. These are:

Direct-Key-Access Files

- Number of records preallocated (if zero, the file is indexed)
- Number of words in each record, including one chaining control word
- Number of elements per record
- Access method number
- Bias value
- Backup method

  TR Traced

  DR Dual-recorded

Direct-Key-Access Indexed or Hashed-Access Indexed Files

- Number of entries allowed
- Access method number
- Bias value
- Backup method

  TR Traced

  DR Dual-recorded

Random or Sequential Files

- Number of words per record
- Number of elements per record
- Backup method

  TR Traced

  DR Dual-recorded

If any of these files are chained, a second line on the listing contains

C.L.F.ffff

ffff Chainer file name

C.L.K.eee

eee Chaining key element name

F.L.C.eee

eee The element that contains the pointer to the first and last record in the chain (control element). This element must be a group element in file ffff (refer to Group-Element later in this section).

Following the file introduction lines is a list of elements, in order of ascending location, within each record. No elements are listed for index files. Each element description line may contain the following fields from left to right.

- Gap or overlap

  GA nn (an unassigned area of nn bits exists before this element)

  OV nn (this element shares nn bits with at least one element above it in the listing)
- Word number of word containing the first bit of the element
- Bit position of the first bit
- Element name
- Element type

  S. Simple

  K. Key
- Storage mode
- Element length given in units of words, characters, or bits depending upon storage mode.
- Read security
- Update security
- File relationship, if element is the key element of an indexed file.

  P.I. ffff

  ffff Primary index file name (element is a primary key)

  S.D. ffff

  ffff Secondary data file name (element is a secondary key)

If any group elements appear in the record, these are listed along with the names of all group member elements at the end of the element list for each file. Also printed at the end of each file description is a summary total of all gap and overlap bits.

DATAMAP CONTROL STATEMENT

The DATAMAP control statement provides the same map of a data base as the M parameter presented previously, but allows the user to produce a map without running DATADEF. The format is:

DATAMAP(p_1,p_2,p_3,...p_n)

p_i can be:

B	Binary data base description file	
	B	Description is on file LGO
	B=fn	Description is on user-assigned file name fn
	absent	Description is on file LGO
L	List file	
	L	List on OUTPUT
	L-fn	List on user-assigned file name fn
	L=0-	No list
	absent	List on OUTPUT
NR	NR	Do not rewind description file
	absent	Rewind description file before producing data map

INPUT TO DATADEF

FORMAT SPECIFICATIONS FOR DATADEF INPUT

Format specifications for the input deck to DATADEF are:

- Sections begin with section line that are terminated by a period.
- All statements within the section are terminated by a period.
- Data lines within a specified section are freefield.
- Comment lines are indicated by an asterisk in column 1.
- Comments in the text are introduced with/* and terminated with */.
- Fields to be processed by DATADEF are from column 1 to 72; DATADEF accepts and lists a 90-character line so that souce lines may contain sequence information in columns 73 through 90.
- Blank lines are bypassed, but are listed.
- Any number may be specified in either decimal or octal notation by placing an appropriate radix symbol immediately following the number. D specifies decimal, and B specifies octal. Decimal is assumed if the radix symbol is absent. For example, length is 12B bits is equivalent to length is 10 bits or length is 10D bits.
- Words must be bounded by separators. Allowable separators are period, comma, or space. Separators may not appear in the words, only between them. Whenever separators are allowed, multiple separators are allowed and considered as one. For readability purposes, any of the separators can be used where a separator is needed, except in the case of lists where periods are not allowed.
- Where periods are shown in the formats in this section (to terminate lists), periods must be used.

NOTATION AND SYMBOLS FOR DATADEF DESCRIPTIONS

The formats given in this section are intended to aid the user in writing statements according to the rules of the language. Editorial conventions used include:

- Material enclosed in brackets [ ] may be included or omitted as required by the programmer.
- Of material enclosed in braces { }, only one of the enclosed items can be chosen; the other must be omitted.
- When a pair of brackets is immediately followed by ... (ellipsis), the material within the brackets can be repeated at the user's option.
- All words printed in capital letters are words that have preassigned meaning to the processor.
- All underlined words are required unless the portion of the format containing them is optional. Such words are key words; if any are missing or misspelled, it is considered an error in the program. These words are not underlined by the programmer when entering the statement.
- All words not underlined may be included or omitted at the option of the user. They are used only for readability. Misspelling such words, when they are included in a format, constitutes an error in the output.
- All words printed in lowercase letters represent information that the programmer is to supply. These words generally indicate the nature of the information they represent.
- Periods, where shown, are essential and must be included by the user.

INPUT SECTIONS

DATADEF input is submitted as four separate and distinct sections: identification, record-description, file-description, and file-relationship.

Identification Section

The identification section specifies the information needed to identify the data base by name and is the first section of input to DATADEF.

```
IDENTIFICATION SECTION.

    DATA-BASE-NAME IS cc.
```

The data base name cc is a two-character mnemonic, specified by the user, which identifies the data base to be defined by the following input. The data base name may not begin with a D or T, and must not be SY, as these characters are used by the data manager and other data base utility programs. The data base name may begin with any other letter and with numerals 0 through 4.

Record-Description Section

The record-description section follows the identification section in the input file to DATADEF. The record-description section defines the specific data fields (elements) within the data base. Besides the element name, this section also specifies the data type, field size, position within the record, and security levels. Default values for these element parameters are preset with the IMPLIED commands. These are explained later in this section. At least one record must be defined in this section, but a record description can be used for more than one file. This means that all records within a file have the same structure. Only one record description must exist for each data file in the data base. The user must not describe index files or trace files.

The general format of the record description section is illustrated in figure 3-1.

```
RECORD-DESCRIPTION SECTION.

    RECORD-NAME IS record-name-1.

    ELEMENT NAME IS eee

                                  { X          LENGTH IS nnn CHARACTER[S]   }
                                  { 9          LENGTH IS nnn CHARACTER[S]   }
                                  { F          LENGTH IS 1 WORD             }
    STORAGE MODE IS               { I          LENGTH IS nnn BIT[S]         }
                                  { D          LENGTH IS 2 WORDS            }
                                  { L          LENGTH IS nnn BIT[S]         }
                                  { R          LENGTH IS 1 WORD             }
                                  { U          LENGTH IS nnn BIT[S]         }

    [LOCATION-WORD IS             { nnnn START-BIT IS nn }                  ]
                                  { eee                  }

    [READ-SECURITY IS n ]

    [UPDATE-SECURITY IS n].

                                  { X }
                                  { 9 }
                                  { F }
    [IMPLIED-STORAGE MODE IS      { I } .  ]
                                  { D }
                                  { L }
                                  { R }
                                  { U }

                                  { CHARACTER[S] }
    [IMPLIED-LENGTH IS nnn        { WORD[S]      } .  ]
                                  { BIT[S]       }

    [IMPLIED-READ-SECURITY IS n.]

    [IMPLIED-UPDATE-SECURITY IS n.]

    [GROUP-ELEMENT NAME eee CONTAINS ee1,ee2,...,een. ]
```

Figure 3-1. General Format of Record Description Section

RECORD-NAME

The RECORD-NAME statement provides a name for the set of elements that comprise the record. One or more element description statements must follow the RECORD-NAME statements in order to supply a format for the record.

record-name-1 — One- to seven-character alphanumeric name assigned by the programmer to the record. It must begin with a letter or a numeral 0 through 4.

ELEMENT

ELEMENT identifies an element to be contained in the record. There must be at least one element in the record (up to 4095 may be specified). Each element name in a record must be unique as this name is kept on the element descriptor table for each data base, and is used as an identifier when creating or manipulating a data base.

eee — Three-character mnemonic assigned to the element by the programmer. This mnemonic must start with an alphabetic character or a numeral 0 through 4.

STORAGE-MODE

This clause specifies the storage mode and field justification of the data base element.

Any string of alphanumeric characters (and '-') may immediately follow the mode character for clarification.

Mode	Description of Mode Allocation
F	An unnormalized floating-point number that is forced to a full machine word (length 60 bits). If the previous allocation did not fill the machine word, the space between the end of the previous element and the beginning of the next machine word is not used. The COBOL 4 equivalent is 9, with a usage of Computational-1. There is no COBOL 5 equivalent.
D	A double precision field that fills two words (fixed length of 120 bits). The convention used here is similar to type F except two machine words are allocated for this element type. No COBOL equivalent exists.
R	A normalized floating-point number that is forced to a full machine word (length 60 bits). If the previous allocation did not fill the machine word, the space between the end of the previous element and the beginning of the next machine word is not used. COBOL equivalent is Computational-2.
L	A logical field that is either true or false (if the bit is set, the condition being tested is true; if clear, false). Minimum length is 0 bits; maximum length is 60 bits. Although only 1 bit is used for this field (the leftmost bit), it may be assigned a longer field length. This element type is always allocated to the next available bit position and if longer than 1 bit, the remaining n-1 bits are allocated to this field without consideration of word boundaries. No COBOL equivalent exists.
I	A signed integer field. Minimum length is 0 bits; maximum length is 60 bits. The method for allocation is: the machine word currently being allocated is examined. If enough space remains within this word to hold the I field, it is allocated to the current word; if not enough bits remain in the current word, the element is allocated to the next word starting in bit 59. In this case, the short field in the current word is not used and the next word becomes the current word. No COBOL equivalent exists.
U	An unsigned integer field. Minimum length is 0 bits; maximum length is 60 bits. The method for allocation is: the machine word currently being allocated is examined. If enough space remains within this word to hold the U field, it is allocated to the current word. If not enough bits remain in the current word, the element is allocated to the next word starting in bit 59. In this case, the short field in the current word is not used and the next word becomes the current word. COBOL equivalent is Computational-1.
X	Display code character(s) that is left-justified within the field. Minimum length is zero characters; maximum length is 680 characters. This element type must begin on a word's character boundary (bit 59, 53, 47, 41, 35, and so forth). The element may start on any characer boundary and is allocated the next nnnn character positions (refer to Length) without consideration of machine word boundaries. COBOL equivalent is X.
9	Identical to X mode except that only sign and numeric characters are allowed and the maximum length is 18 characters. COBOL equivalent is 9.

An explanation of processes used to convert elements from one mode of storage to another appears under Element Processor in section 5.

LENGTH

This optional clause identifies the maximum length of an element by the numeric value nnnn. The units are bits, characters, or words with one character equal to 6 bits, and one word equal to 60 bits. The units cannot be mixed in one statement, but may be mixed in one record.

LOCATION-WORD, START-BIT

This optional clause allows the user to position the data field within the record for most efficient use of storage. If the LOCATION-WORD clause is not specified, DATADEF automatically places the element in the next available position based on the rules of allocation (refer to Storage-Mode).

nnnn Word address within the record; the record is considered to start with word 1.

nn Initial machine word bit position of the entry (59 is leftmost, 0 is rightmost).

The optional form of this clause, LOCATION-WORD is eee, may be used to more flexibly define elements if the programmer wishes to have the elements overlap. The element being defined in the current statement will overlap element eee; in fact, it will have the same location word and start bit as eee.

eee Previously defined element name in the same record.

Defining overlapping elements in this manner results in easier definition maintenance, because changing the record size does not alter the relationship between overlapping elements. Since overlapping elements are allowed, care must be taken to avoid unintentional overlap of data within a record. All elements specified in the record description section may be overlapped, if required.

READ-SECURITY, UPDATE-SECURITY

These two optional clauses specify the security levels for reading and updating elements within a record. The value of n must lie between zero and seven inclusive (with seven being the highest security level). The default is zero. Refer to Security in section 1.

IMPLIED-STORAGE-MODE

This statement presets a default value for the storage mode. This value holds as a default value for all statements and records that follow it. The parameters are explained in Storage-Mode.

IMPLIED-LENGTH

This statement presets a default length for all statements and records that follow it. For an explanation of the parameters, refer to Length.

IMPLIED-READ SECURITY,
IMPLIED-UPDATE-SECURITY

These statements preset a default security for all statements and records that follow it. Refer to Read-Security, Update-Security for an explanation of the parameters.

GROUP-ELEMENT

This statement creates a list of predefined elements which are passed as a group. Since this mode of passing elements does not allow individual conversion information for each element, the elements are passed not more than one per 60-bit machine word. The data format of the grouped elements remains the same as the format in which they were already defined in the data base.

File-Description Section

The general format of the file-description section is:

```
FILE-DESCRIPTION SECTION.

FILE-NAME file-name-1

{{ IS A DIRECT-ACCESS-INDEX WITH nnnnnn ENTRIES
 { IS A HASHED-ACCESS-INDEX WITH nnnnnn ENTRIES
 [ATTACH-MODE IS am]

CONTAINS RECORD record-name-1
[ATTACH-MODE IS am] [key-specification]
```

The optional key-specification has the following format.

```
RANDOM-KEY IS ee1

PRIMARY-KEY IS ee2 [REPEATED nnnnnn TIMES]
      [ACCESS-METHOD IS nn] BIAS-VALUE IS
      +nnnnnn
      -nnnnnn
      nnnnnn
[SECONDARY-KEY IS ee3 [,ee4] ...].
```

All data and index files in the data base must be described in this section.

FILE-NAME

This statement specifies the four-character file name. If either the direct-access indexed or the hashed-access indexed option is specified, this file name indicates which file is to be the primary index to the data file listed in the data-file statement in the file-relationship section. All of the options specified for this statement follow immediately after the file name specified. No intervening punctuation is necessary between options. The file name must begin with an alphabetic character or with a numeral 0 through 4.

DIRECT-ACCESS-INDEX, HASHED-ACCESS-INDEX

The access methods specified in this clause define the procedure used to access the data file specified in the file relationship section. A direct-key- access index is used whenever the access-method algorithm for the data file can ensure unique results for any legitimate key. The hashed-access indexed is normally used whenever the data file's access method cannot guarantee a unique result for every key value. This type of index allows for hashing synonyms and possible index block overflow on ADDR operations. Use of the hashed-access indexed or direct-key-access indexed prohibits use of any of the options otherwise allowed for the file-name statement except attach-mode, because these types of index files are fixed format (refer to section 2).

nnnnnn ENTRIES

The entry count (nnnnnn ENTRIES) indicates the number of records to be preallocated for both types of index files. For a hashed-access indexed file, the number of base index blocks preallocated depends upon this value; however, overflow blocks may be dynamically allocated beyond this preset number of blocks.

ATTACH-MODE

This optional clause specifies the mode in which the user desires the files to be attached by the transaction facility during initialization. The default mode is preset to modify mode by DATADEF. The following modes are allowed.

Mode	Description of Mode Allocation
M MO MODIFY	Allows replacing information within the file or adding information at the end of the file.
R RD READ	Allows read only.
RM READ-AND ALLOW-MODIFY	Read access to a file allowing another user to access the same file in modify mode.
W WR WRITE	File cannot be attached by another user; all users must have completed their operations on the file.

If the procedure file (TAFxxxx) used to bring up TAF attaches the files for the TAF executive (using the NOS ATTACH control statement), the attach mode specified in the procedure file overrides the attach mode specified in the File-Description Section discussed previously in this section. Specify the attach mode in the procedure file because changing the attach mode in the procedure file does not require regenerating the data base.

CONTAINS

This clause associates a record defined in the record description section with a file. The record named must have been previously described. Only one record format is allowed for each data file. However, more than one file may use the same record description if desired to effectively use the same set of elements.

RANDOM-KEY

This optional clause informs the data manager that the user will access this file using a system address. The record description must include an element ee1 defined in the record.

PRIMARY-KEY

This optional clause defines for the data manager which element in the record is the primary key element for indexed or direct-key-access files. This element is the element referenced for all accesses to the record.

REPEATED

This optional clause is intended for direct-key-access files. This clause indicates the number of records to be preallocated to the file.

ACCESS-METHOD, BIAS-VALUE

This optional clause defines the key manipulation method and bias (nnnnnn) in decimal which will be processed by the method to create a logical record number. Care must be exercised to ensure that bias processing by method results in a positive number. Method must be defined within the data manager.

SECONDARY-KEY

This optional clause defines one or more secondary keys used to access this record. A secondary key always leads to the primary key which then is used to access the record. A period must immediately follow the last element name. Keys are further explained in section 2.

DECISION TABLE TO VERIFY FILE DESCRIPTION

The decision table (shown in figure 3-2) can be used to verify the logical accuracy of the file-description statement. The upper portion of the table indicates what combination of specifications may be supplied and the lower portion indicates what functions are performed by the data manager for each of the possible resultant file types.

NOTE

X in the table indicates the feature is applicable to the file type; - indicates it is not applicable.

Specifications in File Description Section		Sequential	Random Files		Indexed Files				Direct Key Access Files			
		1	2	3	4	5	6	7	8	9	10	11
	Primary-key	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y
	Random-key	N	Y	Y	N	N	N	N	N	N	N	N
	Repeated nnnnnn times	N	N	N	N	N	N	N	Y	Y	Y	Y
	Secondary-key	N	N	Y	N	N	Y	Y	N	N	Y	Y
	Access-method	N	N	N	N	Y	N	Y	N	Y	N	Y
ADDR	Pass disk address to user	-	X	X	-	-	-	-	-	-	-	-
ADDR	Record added at end of information	X	X	X	X	X	X	X	-	-	-	-
ADDR	Add at key if slot empty and key ≤ nnnn	-	-	-	-	-	-	-	X	X	X	X
	Primary index maintained	-	-	-	X	X	X	X	-	-	-	-
	Preallocated	-	-	-	-	-	-	-	X	X	X	X
	Secondary-key (not maintained on-line)	-	-	X	-	-	X	X	-	-	X	X
	Key transformed by access-method	-	-	-	-	X	-	X	-	X	-	X
	Simple chain in file	X	X	X	X	X	X	X	X	X	X	X
	Complex chain in file	-	-	-	X	X	X	X	X	X	X	X
	Purge marks record empty	-	X	X	X	X	X	X	X	X	X	X
	Purge causes chain modification	-	-	-	-	-	-	-	X	X	X	X

Figure 3-2. File Types Supported by the Transaction Subsystem

File-Relationship Section

The file-relationship section provides options for defining the interrelationship among certain files in the data base.

- Indexed (data) files may be associated with their respective direct-access indexed or hashed-access indexed files.
- The process of data access via secondary-keys may be defined by associating primary data files with secondary data files.
- Records having some property in common may be established within a file.
- Files may be defined as dual-record files wherein all data written to the prime file is also recorded on a dual copy.
- Files may be defined as traced files, wherein all writes to the file are also recorded sequentially on file xxTFIL for recovery purposes, where xx is the data base name.

The general format of the file relationship section is:

FILE-RELATIONSHIP SECTION

[DATA-FILE IS file-name-1 [PRIMARY-INDEX IS file-name-2] [SECONDARY-KEY ee3 THROUGH FILE file-name-3 HERE ELEMENT ee4 IS PRIME-KEY] ...]]

[CHAIN FILE-NAME file-name-4 BY ELEMENT ee1 WHICH IS THE PRIME-KEY OF FILE-NAME file-name-5 AND USE ELEMENT ee2 FOR CONTROL]

[DUAL-RECORD file-name-1[[,file-name-2]...].]

[TRACE file-name-3[[,file-name-4]...].]

The optional clauses may appear in any order any number of times and need not be grouped by file.

DATA-FILE

This statement defines the data file with which the primary-index and secondary-key statements are associated. It is required only for indexed data files and files which can be accessed by secondary keys.

PRIMARY-INDEX

This statement identifies the index file associated with the data file defined in this section. The access-method stated for file-name-1 in the file description section is used to select the proper intermediate record in the index file when retrieving a record from the data file. The index file must be declared to be a direct-access indexed or hashed-access indexed type file. This statement may occur only once for each data file.

SECONDARY-KEY

This statement describes the relationship of data file file-name-1 and data file file-name-3. Element ee3 is a secondary key of file-name-1, and ee4 is the element in file-name-3 that contains the primary key value for file-name-1. In order to retrieve a record from file-name-1, the access-method stated for file-name-3, if any, is used to select the proper intermediate data record on file-name-3. Element ee4, containing the value of the primary key, is then used to access the index file or the data file if the file is not indexed. The record structure for a secondary data file must be defined by the user. Each secondary key stated for the data file in the file description section must be satisfied in the file relationship section.

CHAIN

This statement defines a relationship between a complex-chained data file and a chainer file. For a description of complex chaining, refer to Chaining and Secondary Keys in section 2. File-name-4 is the complex-chained data file in which ee1 is an element (the chaining element). File-name-5 is the chainer file in which ee2 is a group element (the control element) containing two simple elements. The primary key of file-name-5 must have the same description as ee1, except for the name. The primary key of file-name-4 must have the same description as the simple elements of ee2, except for the name. File-name-4 cannot be a sequential or random file.

DUAL-RECORD

This statement defines a dual copy of each file in the list. File names specified must be four-character names of previously described files. Each dual file is denoted by DATADEF by a seven-character name, the first character of which is a D. The next two characters are the data base name and the remaining four characters are the same as the name of the prime copy of the file. The same file cannot be both dual-recorded and traced.

TRACE

This statement specifies that this file is to be traced. Refer to the description of tracing and trace files in section 5. The same file cannot be both dual-recorded and traced.

DBFORM is the utility program to form the user's data base. At data base definition time, the user supplies a description of all data elements and data files to reside in the data base. DATADEF converts this description to a file called the element descriptor table (EDT) for the data base. DBFORM then uses the EDT to create the user's initial data base. The data base may be later modified, using DBFORM when on-line or batch operation is not hindered.

Once the data base is defined, it is ready for access by the transaction facility. Data may then be entered and retrieved through the use of data manager requests from a task for a batch program.

GENERAL FUNCTIONS OF DBFORM

The following sections provide a brief description of the general functions of DBFORM.

CREATE DATA BASE

DBFORM creates the user's data base. The user first runs DATADEF to create the EDT for the data base. This EDT must reside on a direct-access file and have the same two-character alphabetic file name as the data base.

Before running DBFORM, the user may designate the device on which each file in the data base is to reside. One can either define a file with the DEFINE control statements or use the DEF input directive (refer to Input Directives later in this section) when he runs the DBFORM create option. Some or all files in the data base may be defined by the user; however, passwords may not be used. DBFORM defines a file on the default device if the user has not defined it.

PURGE DATA BASE

DBFORM purges the user's data base. The user specifies the EDT file on the DBFORM control statements, and this file and all the files in that data base that reside on default devices are purged. Any files defined by the user as residing on auxiliary devices must be purged by the user.

REFORMAT DATA BASE

DBFORM reformats the user's data base. A reformat run performs one or more of the following functions.

- Adds one or more files to the data base. DBFORM first compares the existing EDT for the data base with the new EDT supplied by the user, and then adds any files present in the new EDT which are not in the old one. Any or all files to be added can be user-defined as specified in the create option; a dual file can also be defined. A D must be added to the beginning of the file name on the DEFINE control statements or DEF directive for a dual file.

- Purges one or more files from the data base. DBFORM purges all files in the current data base not included in the new user-supplied EDT, including omitted dual copies.

- Reformats elements within a file or files to conform to a new EDT record description. DBFORM converts and/or positions data in a record to allow for changes in an element's format or position in the record. New elements can be added and old elements deleted.

- Lengthens or shortens direct-access files and shortens random files. These file types are shortened only by removing inactive record space from the end of the file. Changing the length of a direct-access file requires a new EDT. Shortening a random file requires an SRF input directive, described under Input Directives.

- Removes inactive record space from sequential and indexed files. Removal occurs throughout the file, not only at the end. This function requires an RIR input directive.

- Writes simple-chained and complex-chained indexed files by chain. This causes all records in a chain to be sequential on mass storage. As a result, chain traversal of complex chains is more efficient. The input directive for this function is WBC.

As indicated, some of the reformat functions require a new EDT. The user must supply the entire data base description with the necessary modifications to DATADEF which creates the new EDT. The new EDT may reside on any file having a legal file name up to seven characters. DBFORM replaces the old EDT with the new EDT on the reformat option. Extreme care must be taken in creating a new EDT, because any files omitted are purged from the existing data base and any omitted elements are removed.

For the reformat, a mass storage or magnetic tape file is also necessary for a dump file. If the user does not specify a previously defined file name, a magnetic tape is assumed and the equipment is requested. [Refer to Description of Run Options (OP=yy) later in this section.]

TRIAL REFORMAT OF DATA BASE

DBFORM may perform a trial reformat of the data base. When the data base EDT changes, the user has the option to simulate the reformat to detect errors which might occur. No dump of reformatted data is performed and no recovery is provided with the trial reformat. The data base is not altered. Conversion of elements, changes in file length, new file names to be added, and so on, are checked and descriptive error messages are issued.

DBFORM CONTROL STATEMENT

The DBFORM utility is called by the user with the following command.

DBFORM (DB=xx,OP-yy,p_1,p_2,....p_n)

xx Any two alphabetic characters specifying the name of the data base (EDT file).

yy Any one of the following run options:

- C Create new data base
- P Purge data base
- R Reformat data base
- TR Trial reformat of data base

p_i Parameters useful for reformat or trial reformat options; any of the following in any order:

D	Dump to DBFORM default file, TAPE
D=fn	Dump to user-assigned file name fn
D omitted	No dump is performed on reformat option if L is present; otherwise, dump to default file, TAPE
L	Load from file assigned to D parameter if present; otherwise, load from TAPE
L=fn	Load from user-assigned file name fn
L=0	No load is performed on reformat option
L omitted	Load from file assigned to D parameter if present; otherwise, load from TAPE
LO	List errors from reformat to OUTPUT
LO=0	No list errors from reformat
LO omitted	List errors from reformat on OUTPUT
N=fn	New EDT on user-assigned file name fn
N or N omitted	Illegal; error if reformat needs a new EDT
R	Recover reformat option from last checkpoint
R omitted	No recovery
I	Input directives are on file INPUT
I=fn	Input directives are on file fn
I-0	No input directives
I omitted	Input directives are on file INPUT

DESCRIPTION OF DATA BASE PARAMETER (DB=xx)

xx represents two alphabetic characters used to identify the EDT for a specific data base. Each file name in the data base consists of six characters, the first two being the data base name. The DB is required on every DBFORM call.

DESCRIPTION OF RUN OPTIONS (OP=yy)

File TCF contains the names of the data bases supported by the TAF data manager in DMS statements that have the TAF data manager specified and the status set to ON. TCF is an indirect access file. After running DBFORM with the create or purge option (with TCF as a local file), use REPLACE, TCF to over-write the old TCF with the new version.

C-Create New Data Base

If TCF is present at the control point, DBFORM checks to see if it contains an entry for xx in one of the DMS statements with the TAF data manager specified and the status set to ON. If DBFORM finds no such entry, or if TCF is not at the control point, DBFORM checks the file names described in the EDT.

If a file is defined but not empty, DBFORM aborts with the message FILE NOT EMPTY - filename. If a file exists as a local or common file type, DBFORM aborts with the message FILE NOT PERMANENT - filename.

Files not already defined by the user are defined according to user-supplied input directives; if no directives are specified, the system default device is assumed (refer to Input Directives). If the user specifies the device by using the DEF directive, DBFORM requests operator assignment of the proper equipment when the file is defined. Space is preallocated for direct-key-access files. If TCF is at the control point, DBFORM makes an entry in TCF for xx of the form DMS(TAF,ON,xx). Files defined, files preallocated, and length of each file are specified on OUTPUT. The control statement specifying the create option for data base AA appears as follows:

DBFORM(OP=C,DB=AA)

Directives are on file INPUT.

P-Purge Data Base

The EDT file for the data base and all files (which reside on default devices) described therein are purged. If TCF is at the control point and contains an entry for xx in a DMS statement of the form DMS(TAF,ON,...), the entry is removed from the statement. If xx is the only entry on the DMS statement, the entire statement is deleted. This removal/deletion is done prior to deleting any of the associated data base files. This procedure allows the user to delete an entry in the TCF file even though the associated data base files might be deleted via PURGE commands. For example, if the data base is on various auxiliary devices, the user would use the PURGE command to delete all of the associated data base files and use DBFORM with the purge option to update the TCF. Various trivial errors will occur on the DBFORM purge run if the data base resides on auxiliary devices. These messages are to be expected because DBFORM only purges files on the default device.

Names of the purged files are specified on OUTPUT. The control statement for purging data base AA appears as follows:

```
DBFORM(OP=P,DB=AA)
```

Input directives are unnecessary for the purge option. Journal files, trace files, and other files associated with the data base, but not defined in the EDT, are not affected.

R-Reformat Data Base

DBFORM performs any or all of the functions listed previously under Reformat Data Base. Either a new EDT must be specified on the control statement, or input directives must be supplied, or both. Otherwise, DBFORM issues the message NEITHER DIRECTIVES NOR EDT GIVEN FOR REFORMAT. A dump and load are required when a new EDT is necessary. The new EDT, reformatted files, and tables are dumped to the file specified by the D parameter during the dump operation. Only those files in the data base which are affected by the reformat are dumped. The load portion of the reformat reads the dump file and adds files, changes their lengths, and so on, according to the tables on the dump file. No change occurs to the data base until the load is executed. The dump and load may be done in separate DBFORM runs.

For RIR, SRF, and WBC directives, DBFORM duplicates the action for dual-recorded files. If a file is shortened, its copy is also shortened, and so forth. The user does not specify the dual file in a separate directive. For the DEF directive, action is taken only on the file specified, not on its copy. This allows the dual files to be on separate devices. To define a dual copy of a file, a user must specify a DEF directive for it. The dual copy file name is a D followed by the name of the primary file.

The DAT input directive is needed for a reformat load unless the user loads form the dump file in the same run as it is dumped. This directive is a precaution against loading from the wrong file. The date given must be the date it was dumped. DBFORM checks this date on the directive against the one written on the dump file and aborts if it is not the same. The date on the dump file is also compared with the current date to prevent loading from a file which is older than a preassigned number of days, determined by an assembly parameter.

The DEF directive requires a new EDT containing the description for the file to be defined as well as the description for the remaining files in the data base.

The RIR, SRF, and WBC directives do not require a new EDT. If no new EDT is specified, the data base files are changed immediately by these directives. If a new EDT is specified, the file or files affected by these directives, as well as files changed by the new EDT, are reformatted to the dump file. Then the function specified by the directive does not actually take place until the load.

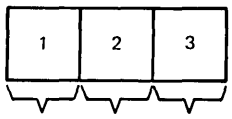
The control statement for reformatting data base AA, therefore, is quite flexible. One of the possibilities is:

```
DBFORM(OP=R,DB=AA,N=NEWEDT,D=DMP)
```

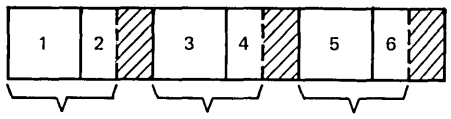
In this instance, the changes to be made to the data base are found by comparing the file NEWEDT with the current EDT on file AA. Files to be changed and tables would be dumped to file DMP and loaded from DMP. At the end of the load operation, file AA would be replaced by file NEWEDT.

CAUTION

Extreme care must be taken when reformatting random files. If the new description causes the records to lengthen and require another sector, the former record on sector 3 would not be on sector 5 and 6, and so forth (if the records were one sector in length before the reformat).



would become:



NOTE

Brackets represent one logical record

TR-Trial Reformat of Data Base

DBFORM simulates the reformat option without writing or loading the dump file or altering the data base. The D parameter is needed for a scratch file. If any errors occur, appropriate messages are issued to OUTPUT and the simulation continues. The control statement for a trial reformat of AA could be:

```
DBFORM(OP=TR,DB=AA,N=NEWEDT,D=DMP)
```

DESCRIPTION OF REMAINING PARAMETERS (p1, p2,...pn)

D-Dump File for Reformat Option

This file can be a magnetic tape or permanent file. The user must have previously defined the dump file if it is permanent, but it need not be at the control point. If it is a magnetic tape file, it can be assigned to a tape unit and DBFORM will find it at the control point. If no D filename is given, the default name TAPE is used and the operator assigns the tape unit when requested. This parameter is ignored if the R or TR run option is not selected.

L-Load File for Reformat Option

If DBFORM cannot find the filename equivalenced to L at the control point or attach it, it assumes it is a tape and requests operator assignment of the tape unit for file TAPE. This file can consist of many reels. An expiration period exists, determined by a DBFORM assembly constant, after which time this file cannot be loaded.

LO

This parameter can be set to zero to prevent element reformat error messages from being issued to file OUTPUT on a reformat or trial reformat run. The default is to list the errors, which is especially useful during a trial reformat to eliminate errors before the actual reformat is performed.

N-New EDT File

A new EDT is required for many types of data base reformats, such as adding or purging files. After a reformat, this file is substituted for the old EDT. This parameter is unnecessary on a data base creation run since the EDT file has the same name as the data base.

R-Recovery Parameters

Recovery of the reformat dump or load operation allows the operator to resume processing after a job interrupt. Without recovery of the dump operation, the user would have to restart the job which could be very time consuming. Without recovery of the load operation, the user's data base would not be intact.

The control statement for recovery must include all the parameters of the interrupted job plus the R parameter. The input directives are not needed for recovery, except for DAT, which is used at the beginning of a load. If the dump file is on tape, the dump or load may be restarted from the begining of the reel being processed when the job was interrupted. Periodically, DBFORM checkpoints on a direct-access file. This file is purged at job completion. If the dump file is on disk, the job is restarted from the last checkpoint on the file. No recovery is provided from the beginning of the dump file on the dump operation; the job must be restarted from the beginning.

For example, if it is necessary to deadstart while DBFORM is in the middle of a reformat on data base AA, DBFORM(OP=R,DB=AA,D=DMP,L,N=NEWEDT,R) could be entered. The dump or load would be restarted from the last checkpoint, whichever operation was in progress before termination.

I-Input Directive File

DBFORM assumes INPUT unless otherwise specified. The file must be at the control point and positioned to the record containing the directives. The purge option assumes no input directives.

INPUT DIRECTIVES

Input directives are allowed on all run options except purge data base, OP=P. Slashes and commas are required where shown. Embedded blanks are not allowed. All fields are required except ee. The file name must be six characters unless it specifies a dual file, in which case it is seven characters, the first character being a D.

/DEF,filenam/DEV,xx,ee	Define file filenam on device type xx, equipment number ee. xx can be: DI 884-21 Multiple Disk Drive

If a procedure file defines the files (using the NOS DEFINE control statement), the device residency specified in the procedure file overrides the device residency specified on the DBFORM control statement. It is recommended that the user specify the device residency in a procedure file because changing the device residency in a procedure file does not require regenerating the data base.

/SRF,filenam,nnnnnnnn	Shorten random file filenam to nnnnnnnn records.
/RIR,filenam	Remove inactive records from file filenam. Then, rewrite the file by chain. Legal only for sequential or indexed files.
/WBC,filenam	Write file filenam by chain. Illegal for complex-chained random and complex-chained direct-key-access files. This directive causes DBFORM to remove inactive records from complex-chained files.
/DAT,yy/mm/dd	Creation date of load file, needed on a load if dumping and loading different dump files or in different runs. yy=year, mm=month, dd=day.

When specifying equipment number, ee must be the equipment status table ordinal associated with the device on the E display.

DBFORM ERROR MESSAGES

Three types of error messages are issued to file OUTPUT by DBFORM.

The first type consists of fatal errors issued to file OUTPUT by the input directives syntax checker. All directives are checked initially by DBFORM. If fatal errors exist, error messages are issued and DBFORM aborts.

The second type consists of the error messages which result from an error in reformatting records. These are explanations of errors received from the element processor which DBFORM uses for retrieving, converting, and positioning elements. They are issued to the file OUTPUT and are not fatal.

The third type of error message is issued to the dayfile.

Input directive error messages and dayfile messages are described in appendix D. Element processor error messages are described in the following paragraph.

ELEMENT PROCESSOR ERROR MESSAGES

The following list of element reformatting errors on file OUTPUT is preceded by a list of possible error codes and a brief description of each.

Error Code	Description
1	User's data overflows data base field
2	Attempt to convert type 9 field that is longer than 18 characters
3	User parameter list error
4	Illegal data type code
5	Illegal conversion request
6	Nonexistent element requested
7	Integer too large for 10-character alpha conversion
30	Group elements not defined properly in EDT
11	Security violation

The error list is in the following format.

SEQUENCE NUMBER	ERROR CODE	DATA BASE
XXXXX	XXXXX	XX

FILE NAME	ELEMENT DESCRIPTOR
XXXXXXX	XXX

This information is intended to enable the user to pinpoint where the error occurred.

INFORMATIVE MESSAGES

Informative messages are issued to the file OUTPUT. Errors associated with these messages, if any, are not fatal.

The following header is included at the top of each page on which these messages appear; it identifies the file data included in several of the messages.

FILE NAME	FILE TYPE	OLD NO. RECORDS	
NEW NO. RECORDS	RECORD LENGTH	DEVICE TYPE	EQ NO.

Record length, when shown, is in central memory words. The notation filedata in a message format indicates that the message includes this information. The messages are described in appendix D.

DATA MANAGER COMPONENTS

The data manager can accept and process requests for information conversion, storage, and retrieval. The data manager is divided internally into the following four major components.

- A control program
- A file access processor
- A series of function or command processors
- An element processor

CONTROL PROGRAM

The control program maintains a buffer status table (BST) that contains one entry for each active data base request that is present in the data manager at any given time. Although data manager buffers and the BSTs associated with these buffers are not directly accessible to the user. the information contained therein is accessible to users through the data base access (DBA) commands presented in this section. The control program receives DBA requests from the transaction executive, interprets each request, and guides the interaction among the element processor, function processors, and the file access processor. The control program is also responsible for controlling the allocation of central memory buffer space and the issuance of input/output requests.

The central memory buffers are not accessible by user programs. They are reserved for input/output optimization by the data manager. Each executing user program is allowed a certain number of buffer areas (determined by NREC,† an assembly constant) for each file that it must access if not in block mode. Each buffer is associated with one record from a particular file. The central memory buffers enable the data manager to hold these records in memory while allowing many user programs simultaneous access to them. Conversely, the data manager may lock a record during the time it is in a buffer to prevent other users from accessing it simultaneously in write mode. A user is in block mode on this file after executing a BLKGET command, and may have several consecutive records in this file locked in central memory buffers. A user leaves block mode by accessing a record outside the block in the same file.

On every read request, the data manager checks its buffer areas to determine if the record is already in central memory. This procedure prevents unnecessary disk accessing, since records are not replaced on the disk unless a forced write option was included in a command (refer to Basic Function Processor Options in this section). If a read request is subsequently executed for a different record from the same file, the data manager determines if NREC records already reside in core for this file. Only if NREC records already reside in core is the oldest record (the first one accessed) removed by an internally generated forced write, if necessary, before the new data is placed in the buffer.

NOTE

If the user holds more than one record from a file (that is, NREC is greater than 1) and issues a command that does not use a key, such as GETNR, then the last record into memory is used as the positioning record.

Assume that five records are allowed (NREC=5) and the following sequence of reads occured.

1st read	Record 1
2nd read	Record 3
3rd read	Record 6
4th read	Record 9
5th read	Record 10

If the next request is a GET for record 1, then no disk access is necessary since the record 1 is in memory. However, if the next request is a GETNR (get next record), record 11 (10+1) will be read since record 10 was the last record read into memory.

When each transaction can hold only one record (NREC=1), deadlocks can occur when performing get next or previous record requests (GETN, GETB, GETNR, and GETRB) unless the transaction locks the record before requesting another record. The following sequence illustrates such a deadlock.

1. Transaction A gets record 1 without lock.
2. The transaction subsystem reserves a buffer for record 1 and links transaction A to the buffer.
3. Transaction B gets record 1 without lock.
4. The transaction subsystem finds record 1 in central memory and links transaction B to the buffer.
5. Transaction A performs a GETNR to obtain record 2.
6. With NREC equal to one, the transaction subsystem attempts to release record 1 but cannot because transaction B is using record 1. Transaction A is put into a wait state until record 1 is free.
7. Transaction B performs a GETNR to obtain record 2.
8. With NREC equal to one, the transaction subsystem attempts to release record 1 but cannot since transaction A is linked to record 1. Transaction B is put into a wait state until record 1 is free.
9. A deadlock exists for transaction A and transaction B.

† The released value of Assembly Constant NREC is 1.

These deadlocks can be avoided by locking the record before performing the get next or previous record or setting the NREC installation option to a value greater than 1.

Requests flow through the data manager in this sequence:

1. Programs such as the transaction executive present requests to the data manager through an input queue.

2. The data manager control program removes the requests from the queue and passes them to the appropriate function processors.

3. The control program then gives control of the CPU to the function processor, which then satisfies the request.

4. When the function processor requires an access to mass storage or some other additional system resource, the CPU is returned to the control program.

Whenever a function processor completes a request, it makes an entry in the output queue which indicates to the transaction executive, or other driving program, that the request has been completed. The output queue, like the input queue, is supplied by the driving program; that is, the transaction executive or the batch data manager executive, BDMI. This method of CPU sharing requires that all function processors be reentrant, since any given request may not be satisfied on one pass.

FILE ACCESS PROCESSOR

The file access processor provides the input/output techniques needed for efficient accessing of the various file defined by the user at data base definition time. The file access processor is responsible for transferring data between the central memory buffer areas and the mass storage devices. The processor is also responsible for reporting error codes to the user programs when there is hardware failure.

FUNCTION PROCESSORS

The function processors are responsible for maintaining system integrity and file security while processing data base requests. A brief discussion of function processor usage is given in the following paragraphs. A more detailed description, including calling formats, appears in section 6.

Basic Function Processors

Five basic function processors are provided by the data manager for manipulating the data base. They are ADD, GET, PUT, PURGE, and REPOS.

The PURGE processor operates on a record basis only, while the ADD, GET, and PUT processors can be used to access either records or individual elements or element groups. In adddition, the GET and PUT processors can access a continuous block of records.

The REPOS processor positions any file at either the beginning or end. Thus, it is essentially the same as a rewind or a skip to EOF. For complex-chained files, the REPOS processor can position the file to the beginning or end of the chain. (The chain is defined by its first and last record, the keys of which are contained in the chainer record.†) In both cases, a record or element request must follow the REPOS in order to obtain data.

Basic Function Processor Options

Various options can be used with the GET and PUT processors to facilitate record updating, file security, and data retrieval, as well as to provide for system integrity and speed of operation. The options are referred to as lock, next, put-in-place, block accessing, and forced-write.

The lock option is intended for use during updating. Its use guarantees that the record being accessed cannot be changed by another user until the first user has released the record.

The next option is intended for use in conjunction with REPOS, and enables the user to access all records in a chain without knowing or specifying the actual record keys. Use of the next option results in the retrieval of the next active record in the chain. All nonkeyed types of requests, such as GETN, PUTI, and so on, act on the record last read into central memory for a particular file.

The put-in-place option is intended to supplement the next option. It provides the capability of writing a record on a file without specifying the actual key. Thus, with proper use of these three processors, a user can access the records in a particular chain, one at a time, until he finds a record that he wishes to update. He can then update that record and write it back on the file without being familiar with the file's structure.

The block accessing option for the GET and PUT processors allows users to read and write contiguous groups of records. Its intent is to optimize data base accessing when a large number of records must be processed within a file. After a BLKGET is processed, and a block of records for a file is held in core for a user, the user's next request for this file, other than a BLKPUT, must be of the type that requires a key (GETL, PUTR, etc). A BLKPUT command is legal only if a BLKGET command has been made on this file by this user and the number of records to be written does not exceed the number of contiguous records held by the data manager starting at the record specified on the BLKPUT request. The first record on the BLKPUT must be the same as the first record on the BLKGET.

†For more information, refer to Chaining and Secondary Keys in section 2.

The forced-write option is associated with the PUT function processor and allows the user to force a write on a file at the time the command is issued. When the option is not specified, the data to be PUT is changed in central memory only, and the updated record is not physically written on the file until the end of the transaction or until the user accesses more than NREC records for a particular file. This delay in the physical record transfer is intended to optimize disk accessing and allows many users to have access to a record while the data manager has to maintain only one copy of the record in central memory. The forced-write option allows the user to bypass this delayed-record transferring feature of the data manager and to force the data manager to release the buffer associated with the particular record written. However, forced writes on either the prime or the dual copy of a dual-recorded file does not cause the buffer to be released. A write on both the prime and dual copy causes the buffer to be released.

Additional Function Processors

In addition to the preceding five basic processor, the following processors are provided by the data manager.

UNLOCK

LOCKF

UNLOKF

RELES

RELESAL

CEASE

RECHAIN

DMSTAT

READEDT†

The UNLOCK processor enables the user to free a record for the use by others when he no longer needs it. This feature helps reduce disk accesses by keeping a record in memory where, if it is not locked, it is available to many users simultaneously.

The LOCKF processor enables a transaction to lock out a file such that the whole file becomes inaccessible to other transactions until the locking transaction has completed. This lock applies only to those commands which lock records within this file, such as write commands. Thus, other transactions are able to read the file while it is locked, but are not able to write on it.

The UNLOKF processor is intended for use with LOCKF. It merely resets the status of a file to an unlocked condition so other transactions can have access to it.

The RELES proccessor allows a user to release any buffer areas in memory which are used by the task. This function is intended for use when a program determines that it has the wrong records or the wrong information in certain records, and would like to try to recover. By issuing a RELES comand, the program can free its buffer areas for other use by the data manager without causing any records to be written on the files. RELESAL releases all buffer areas associated with the task.

The CEASE command is the normal method for causing program termination. It can be issued with a request for either normal or abnormal termination. If normal termination is requested, all records which have been modified by the program are written out on their respective files. If abnormal termination is requested, no record transfers occur. Thus, in the case of abnormal termination, the CEASE function is quite similar to the RELES.

RECHAIN causes a record in a chained file to be purged from one chain and inserted into a new chain. An example of its use would be as follows: the data base is set up such that customer records are defined as belonging to certain branches; thus, the customer record file is a complex-chained file.†† One customer decides to move his place of residence so that his account belongs to a different branch office. To transfer his account to the chain belonging to the new branch office, the RECHAIN function can be used.

The DMSTAT processor allows the user to access the 12 BST bits and the data manager record buffer for each record assigned to the requesting task chain. The DMSTAT function returns the file name, BST status bits, and key value (two words in the form specified in DATADEF) in a three-word entry called a record status entry (RSE) for each record assigned to this transaction.

The RSE format is shown in figure 5-1.

The user may specify in the call the maximum number of RSEs to be returned. The DMSTAT function returns the number of RSEs found up to the maximum specified by the call.

COBOL and FORTRAN Extended users may optionally specify in the call from 0 to 11 status bits and their corresponding status bit buffers. The DMSTAT function then examines the specified bit in each returned RSE and indicates its set (1) / not-set (0) condition in the appropriate status bit buffer. Unnormalized floating point (Computational-1 in COBOL) 1/0 indicates the set/not-set condition. Refer to appendix A for an example of a DMSTAT call and an explanation of the BST status bits.

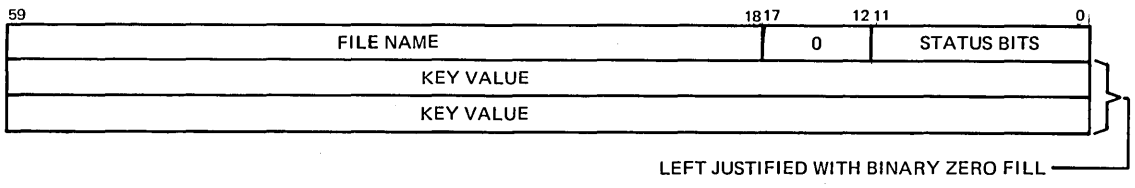


Figure 5-1. Record Status Entry Format

†An installation-specified option must be enabled before this function processor can be used (the default is disabled).
††For more information on complex-chained files, refer to Chaining and Secondary Keys, section 2.

The READEDT processor allows the user to obtain the element descriptors for a specified data base and file from the EDT.

In the first word, the READEDT command returns the number of element descriptors contained in the EDT for the specified data base and file; in the second word, the command returns the number of element descriptors returned; and in the remaining words, the command returns the element descriptors themselves. The first two words are returned as unnormalized floating-point numbers.

Element descriptors for simple or key elements are in the format shown in figure 5-2. Element descriptors for group elements are returned in the format shown in figure 5-3.

ELEMENT PROCESSOR

The element processor consists of a set of routines that enables conversion of individual data elements to a specified type or format as they are being transferred between central memory buffers and user programs. The user, therefore, can define his elements to reside on mass storage in one format while the programs can request the same elements in a different format. Figure 5-4 specifies the legal element conversion for each of the eight element types supported by the data manager. A blank indicates that conversion is legal; the word NO indicates that the conversion is illegal. The element types are described under Storage-Mode in section 3.

In addition to the eight element types used in the data base definition, a type A specification may be used with certain data manager requests that process elements. The type A specification instructs the element processor to get or put an element from or into the data base file according to the type specified in the EDT; thus, no type conversion is specified.

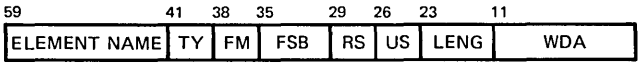
The type A specification must not be used for key elements, but can be used for all other simple or group elements. It is valid with the following requests.

ADDR	GETL	GETT	PUTI
GETB	GETN	PUT	PUTIF
GETBL	GETNL	PUTF	RECHAIN

A description of the most frequently used conversions can be found in appendix B.

FILE BACKUP

Three techniques are available for the file backup: dual-recording, tracing, and pooling. The dual-record technique requires that duplicate files be maintained. The trace technique involves recording each data base alteration to enable file reconstruction. Pooling is accomplished by maintaining an error file. Records are written and pooled on the file if they cannot be restored to their original residence on disk due to hardware failure. A fourth backup technique, called journaling, is discussed in the reference manual.



element name — Three-character name of the element described

ty — Type of element. Possible values are:

0	Simple element
1	Group element
2	Key element

fm — Form of the element in storage (storage mode). The value of fm determines the storage mode as follows:

fm	mode
0	9
1	X
2	U
3	R
4	L
5	I
6	F
7	D

Section 3 contains a description of storage modes.

fsb — First bit of the element (from 59 to 0)

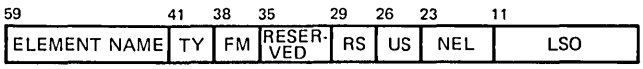
rs — Read security (refer to section 3)

us — Update security (refer to section 3)

leng — Length of the element in bits

wda — Word address within the record in which the element is found; the record is considered to start with word 1.

Figure 5-2. Simple or Key Element Descriptor Format



element name — Three-character name of the element described

ty — Type of element. Possible values are:

0	Simple element
1	Group element
2	Key element

fm — Form of the element in storage (storage mode). The value of fm determines the storage mode as follows:

fm	mode
0	9
1	X
2	U
3	R
4	L
5	I
6	F
7	D

Section 3 contains a description of storage modes.

rs — Read security (refer to section 3)

us — Update security (refer to section 3)

nel — Number of elements in the group

lso — List ordinal. This is a pointer to an internal list of elements in the group. To obtain more information about these group elements, the user should run a DATAMAP (refer to section 3) against the EDT.

Figure 5-3. Group Element Descriptor Format

FROM \ TO	F	D	R	L	I	U	X	9
F								no
D							no	no
R								no
L		no					no	no
I								
U								
X	no	no	no	no	no	no		no
9	no	no	no	no	no	no		

Figure 5-4. Element Conversion Legality

DUAL-RECORD

Dual-recording is provided if the dual-record option is selected for individual files during definition of the data base by DATADEF. For every file where dual-recording is selected, two identical files are created. The prime file's name is the six-character name of the file made up of the two-character data base name followed by the four-character filename. The name of the dual copy is the prime file's name preceded by D, giving a seven-character name.

Full control of dual-recorded files is given to the user, allowing a task to specify which copy of a record it wishes to access for both read and write commands. On read commands, a transaction is able to specify whether it wants the prime copy, dual copy, or either copy, while on write commands, it can specify prime, dual, or both copies. Indication of the record copy desired is transferred to the data manager via the low-order 3 bits of the return flag parameter word as shown in the following table.†

No Recall	Recall	Read	Write
000	001	Read either	Write both
010	011	Read prime	Write prime
100	101	Read dual	Write dual

†The return flag is described in more detail under Command Parameter Definitions in section 6.

This feature provides the use with enough flexibility in the area of file manipulation to enable him to determine whether a transaction has run to completion or not. By employing some appropriate sequence of writing prime records and dual records during an update, the user is able to make an after-the-fact comparison of the two records. Based on the results, he can determine whether his update transaction ran to completion and intitiate corrective action, if needed.

NOTE

If the user wishes to write to either the prime record or the dual-record but not both, one of the forced put requests must be used. All other commands that modify records, such as add, purge, rechain, and all the delayed put requests, modify both prime and dual-records.

TRACE FILES

There is one trace file for each data base. The filename is xxTFIL where xx is the relevant data base (for example, if the data base is named LC, the associated trace file is defined as LCTFIL). Each trace file is a direct-access permanent file residing under the same user number as its associated data base. It may be defined by operations personnel so that device residency may be specified. In interactive mode, the TAF data manager checks at initialization time to see if a trace file exists for each active data base. For those data bases without a trace file at the control point,† it tries to attach the necessary trace files in modify mode. If the data manager cannot find the files, it defines new files which then reside on a default device type. The default device type is an assembly constant in the data manager.

Each time a transaction issues a data base request that causes a write on a traced file, one record is written to the appropriate xxTFIL. The user may turn on or turn off the tracing of selected records for any traced file in the user's data base through the ONTRCE or OFFTRCE commands. The correct trace file is determined by the data base name contained in the transaction terminal validation file for each terminal name (refer to Security, section 1). Files to be traced are specified at data base definition time.

Should hardware problems occur and cause data to be lost, the trace file may be used to recreate the files in question. In this case, a copy of the file from the previous day may be updated through use of the trace file. Because this activity is considered an application activity, a utility program to perform this function is not provided. This method of backup differs from dual-recording in that only a single day's activity is recorded on the trace file, while duplicate copies of all records are maintained by dual-recording.

Trace files are written in variable-length record format (figure 5-5). Each record contains a seven-word header describing the entry. Figure 5-6 illustrates the format of the trace file header. These files are not meant to be accessible from a task at the time they are being written. However, they can be analyzed off-line.

POOL FILES

Additional data base reliability is provided by using error recovery pool files. Figure 5-7 shows the error recovery pool file structure. A pool file exists for each data base. The file naming convention is xxERPF, where xx is the data base name. Operations personnel may define the file to reside on any mass storage device. In interactive mode, the TAF data manager checks at initialization time to see if a pool file exists for each active data base. For those data bases without a pool file at the control point , it tries to attach the necessary pool files in modify mode. If the data manager cannot find the files, it defines new files which then reside on a default device type. The default device type is an assembly constant in the data manager. xxERPF must be a direct-access permanent file under the user number associated with the relevant data base. A pool file contains copies of records which could not be written to their original place of residency because of a mass-storage error.

All records written to an error recovery pool file (ERPF) are accounted for in the copies record address table (CRAT) kept in central memory. Each CRAT entry is one word in length with two 30-bit fields containing the original file disk address and the pool file disk address.

Upon receiving a record request for a file containing one or more entries on the ERPF, the data manager searches the CRAT to determine whether the requested record has been previously pooled, indicating a faulty original file. If it has been, the data manager retrieves the record from the ERPF rather than the original file. If the desired record has not been pooled, it is retrieved from the original file. When a pooled record has been modified, it is rewritten on the pool file rather than on the original file.

Besides acting as an index to pooled records, the CRAT functions as a controlling device which enables the data manager to initiate automatic idle-down procedures. This purpose is served in the following manner.

- CRAT is allocated enough memory to hold 120 entries via an assembly constant.
- At such time as the 100th entry is made in CRAT, the data manager requests the system to begin idle-down (refer to the K.IDLE command in the reference manual).

The CRAT exists in central memory only. It is not written to a file. This avoids the necessity of updating such a file every time a change or addition is made to CRAT. Figure 5-8 illustrates the ERPF structure and format of the header for each entry.

†The files may have been attached by the TAF procedure file (TAFxxxx). The author of TAFxxxx must ensure that the files are attached in a mode that allows TAF to write on them.

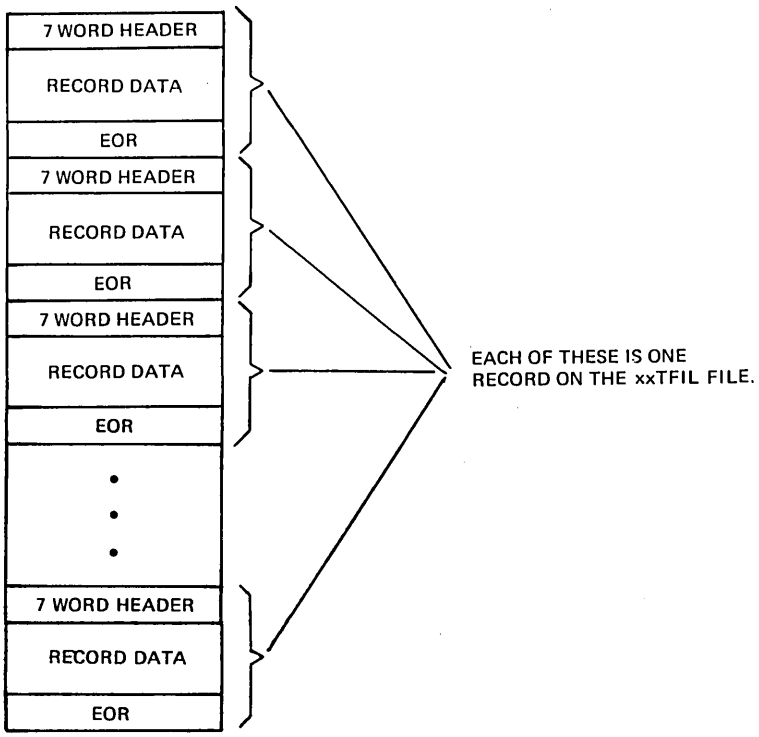
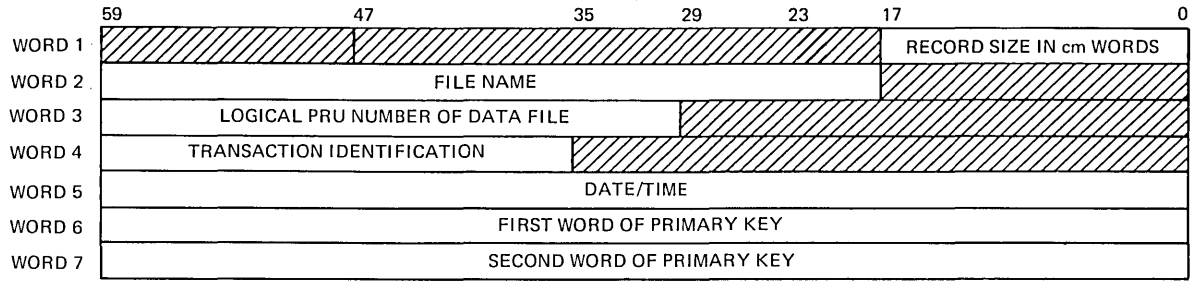


Figure 5-5. Trace File Structure



Word 1	Record size (octal and right-justified, does not include 7-word header)	Word 5	Packed date/time tracing took place (operating system PDATE macro)
Word 2	File name	Word 6	First word of primary key
Word 3	Relative disk PRU of record on data file	Word 7	Second word of primary key
Word 4	Transaction identification (sequence number)		

Figure 5-6. Trace File Header Format

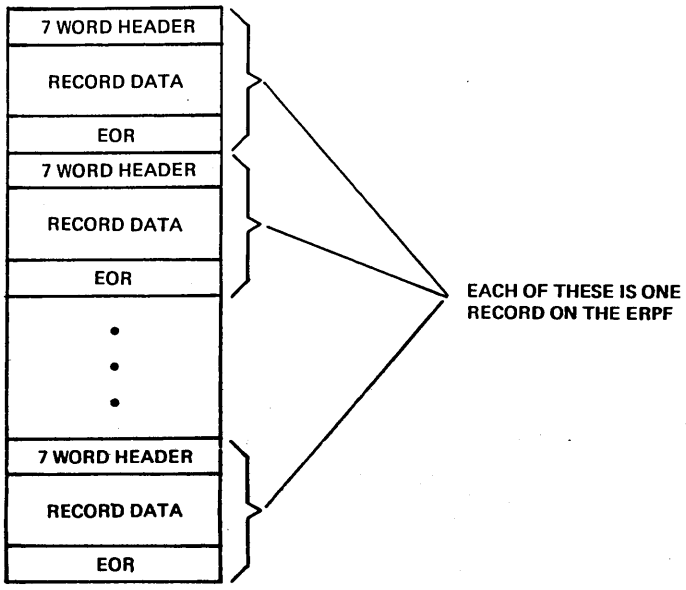
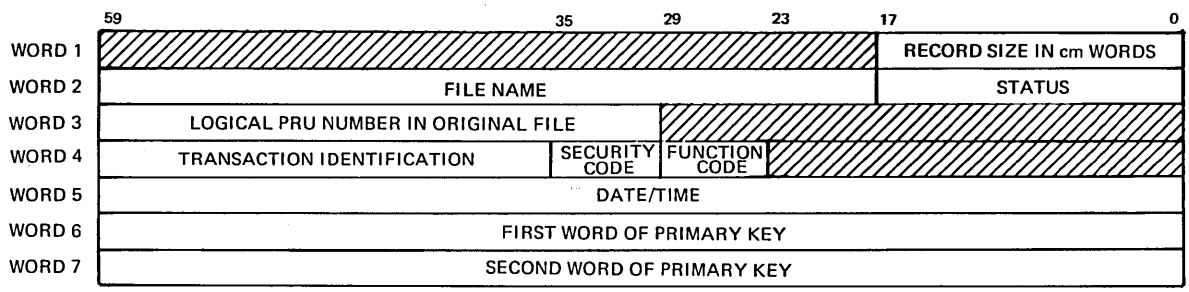


Figure 5-7. Error Recovery Pool File Structure



Word 1 Record size (octal and right justified, does not include 7-word header)

Word 2 File name and value of status word at time of entry

Word 3 Relative disk PRU

Word 4 Transaction identification (or sequence number) (upper 24 bits); security code (6 bits); function code (6 bits)

Word 5 Packed date/time entry was written (operating system PDATE macro)

Word 6 First word of primary key

Word 7 Second word of primary key

Figure 5-8. Error Recovery Pool File Record Header Format

This section describes the function of each data manager request and the calling format of the request for COMPASS, FORTRAN Extended, and COBOL. All requests in COMPASS are in the same format.

fnc addr, recall, f

fnc is the name of the request, (ADDR, GET, etc), addr is the address of the argument list, recall is present if auto recall is selected, and f is any nonblank character and is required. It denotes that addr is a list of addresses. Any alphanumeric character in the position of the recall parameter indicates that the option is selected.

The presence or absence of recall overrides the value of the return parameter as far as auto recall is concerned. However, recall is not used to control dual file access as is the return parameter. The parameters whose addresses are required in the argument list of each COMPASS data manager request are given with the following function call descriptions. The parameters are listed in the order in which they must appear, with one parameter address per word; the list is terminated by a zero word.

COMMAND PARAMETER DEFINITIONS

The following parameters appear in the data manager commands that follow in this section. They are defined here to promote easier understanding of those commands.

Parameter	Definition
chain area	Address in the user area which contains the chaining element value.
	COBOL Computational-1
	FORTRAN Extended Integer
	COMPASS Integer
chain descriptor	Description name (three characters) associated with the chaining element area
count	Positions the record pointers to the beginning or end of the file. Positive number denotes reposition to the end of file; negative number denotes reposition to the beginning of the file. The parameter types are:
	COBOL Computational-1
	FORTRAN Extended Integer
	COMPASS Integer
data base name	Name of the data base on which the file exists (two characters).
EDT address	First word address of the EDT. This buffer must not start in locations 0 or 1 of the task field length. All output from the READEDT function is returned to this buffer in the form stated in Additional Function Processors in section 5.
	COBOL Do not call READEDT from COBOL, because of the difficulty of manipulating the element descriptors in COBOL.
	FORTRAN Extended Locations 1, 2 Floating point
	Location 3, ...Integer
	COMPASS Words 1, 2 Floating point
	Word 3,... Integer
EDT buffer size	Specifies the number of words in the EDT buffer. At least two words must be specified. At most, EDT buffer size minus 2 descriptors are returned. If 2 is specified, no descriptors are returned.
	COBOL Computational-1
	FORTRAN Extended Integer
	COMPASS Integer

Parameter	Definition
element area/data area	First word address in the user area of the record (data area) or parts of the record (element area) which are to be extracted from or inserted into the file (4095 characters maximum).
	COBOL Any valid data type †
	FORTRAN Extended Any
	COMPASS Any
element descriptor	Name assigned to the element area (three characters).
file name	Name of the file to access (four characters).
key area	Name in the user area of a key value associated with the record (20 characters maximum).
	COBOL Any valid data type
	FORTRAN Extended Any
	COMPASS Any
key buffer area	Address in the user area where the data manager will return key values of the records for a block read.
	COBOL Any valid data type
	FORTRAN Extended Any
	COMPASS Any
key descriptor	Element name (from DATADEF) assigned to the key area (three characters).
length	Identifies the maximum length of the data. The units are in characters regardless of data type; 6 bits equal one character. The parameter types are:
	FORTRAN Extended Integer
	COMPASS Integer
number	Maximum number (greater than 0) of record-status entries (RSE) to be returned in RSE buffer.
	COBOL Computational-1
	FORTRAN Extended Integer
	COMPASS Integer
number of records	Number of records in block of records read/write
	COBOL Computational-1
	FORTRAN Extended Integer
	COMPASS Integer
return flag	Indicates if the user wants to stop and wait for the data manager (RECALL), or wishes to continue execution before the data manager operation is complete (no RECALL); also controls dual file accessing.
	COBOL Computational-2
	FORTRAN Extended Real
	COMPASS Floating-point
RSE buffer	Record-status entry buffer. The length of the RSE buffer must be 3*number+1. The number of RSEs returned in the RSE buffer is written in the first word of the RSE buffer. This value also is unnormalized on floating point.

NOTE

The number of RSEs returned in word 1 of the RSE buffer is always ≤ number, as specified in the DMSTAT call. (Refer to appendix A).

COBOL	Word 1 Computational-1
	Words 2, 5 Display code
	Words 3-4, 6-7 Same type as key value
FORTRAN Extended	Integer
COMPASS	Integer

†Refer to Special Considerations for COBOL 4 and COBOL 5 Data Types in this section.

Parameter	Definition
status bit	Identifies that the status is desired for this bit (0 ≤ status bit ≤ 11). (Refer to appendix A.) COBOL Computational-1 FORTRAN Extended Integer
status bit buffer	fwa address in user program to receive the set/not-set condition of the corresponding status bit. The length of this buffer must be ≥ number. COBOL Computational-1 FORTRAN Extended Integer
status word	Word used to return to the user any error status which might arise. (Possible statuses are described in Error Processing, later in this section.) COBOL Computational-1 FORTRAN Extended Real COMPASS Floating-point The status word for COBOL calls should have a value of zero. It must not be blank-filled, as this generates erroneous nonpreventive errors (refer to Error Processing). If the same status word is passed on several calls, it is possible for nonpreventive errors to be passed in the status word from one call to the next.
type	Identifies the format of the data as it exists in the user's area. A Retrieve or write element as specified in the EDT (illegal for key elements). F Floating-point unnormalized (one word). R Floating-point normalized (one word). D Floating-point double-precision (two words). L Logical (0 through 60 bits, leftmost bit indicates true if set, false if clear). I Signed integer field that is right-justified within the word. U Unsigned integer field that is right-justified within the word. X Display code left-justified (string of 6-bit characters). 9 Numeric characters left- justified within the field. All nonnumeric parameters should be left-justified within the word. Types I, L, and U are right-justified in the word with the length less than or equal to 60 bits. (Refer to Storage-Mode in section 3 for a more detailed description of data storage types.) The COMPASS and FORTRAN Extended command formats for record level commands require a type entry for the data area. This entry must be present and must be a valid type code, but it need not relate to the EDT entries for elements contained in the record.

SPECIAL CONSIDERATIONS FOR COBOL 4 AND COBOL 5 DATA TYPES

While the data manager provides for eight different element types, COBOL 4 and 5 process only four types of data: Computational-1, Computational-2, display code, and numeric (display code). These four COBOL types correspond to data manager types as shown in table 6-1.

Type R (floating-point normalized) requires special handling in COBOL calls. The element length and type are not required in calls when the COBOL type is one of the first three in table 6-1, but the length and type are required when the type is Computational-2. For example, to put a real number into a file in a data base, the following calling sequence is required.

ENTER PUT USING filename database statusword returnflag keydescriptor keyarea edl eal ell etl.

edl Element descriptor (display code)

eal Real value to be placed in the file (Computational-2)

ell Length of the real number in characters (Computational-1, value 10)

etl Element type (left-justified display code with a value of R or A)

While there are no COBOL forms that are translated by the data manager into types F, I, or L, elements of these types may be handled in COBOL using the preceding scheme of specifying the element area as Computational-2 and explicitly stating the element length and type. The COBOL user is solely responsible for the manipulation of these data types in his own program.

For parameters or deck values that require true binary zero, then a USAGE IS COMP-2, VALUE IS 0. clause is needed.

In reference to table 6-1, it is possible to write some data types, noted as having no COBOL equivalent, to a data base using FORTRAN and then attempt to read them using a COBOL task. COBOL does not detect the incompatible data type until the task attempts an arithmetic or move statement. At this point the task aborts.

COMMANDS

The purpose and specific rules governing each command and the formats of the command for COBOL, FORTRAN, and COMPASS are given in the following text. All data manager commands should be checked with the STATCHK command which is described in this section.

TABLE 6-1. DATA TYPES

Data Manager Data Type		COBOL 4 Data Type		COBOL 5 Data Type	
Type	Description	PICTURE IS	USAGE IS	PICTURE IS	USAGE IS
X	Alphanumeric (display code)	X(n)	-	X(n)	-
9	Numeric (display code)	9(n)	-	9(n)	-
U	Unsigned integer	9(n)≤15	Computational-1 (positive values only)	9(n)≤14	Computational-1 (positive values only)
D	Double precision floating-point	No COBOL 4 equivalent	-	No COBOL 5 equivalent	-
R	Normalized floating-point	9(n)	Computational-2	No PICTURE clause is used	Computational-2
L	Logical	No COBOL 4 equivalent	-	No COBOL 5 equivalent	-
F	Unnormalized floating-point	9(n)	Computational-1	No COBOL 5 equivalent	-
I	Signed integer	No COBOL 4 equivalent	-	No COBOL 5 equivalent	-

ADDR

This command creates a new storage in a file. The key element is written to mass storage immediately (essentially a forced PUT of the key element). Elements of the record can be added at the same time. The record is allocated on mass storage and the appropriate chain links are made. Any elements specified in the call are added initially only in the buffer area. If the record was purged previously or preallocated, the purge bit is cleared, indicating that the record is active. Both the prime and dual copies are written if the file is dual-recorded. At the completion of the request or when NREC records are added, the remainder of the specified elements are written to mass storage.

NOTE

If the ADDR is being done on a complex-chained file, the first element descriptor in the call after the key parameters must be the chain descriptor (the element in the chained file which is the primary key of the record in the chainer file to which this record is to be chained). The first element area in the call must contain the value of that key (chain area).

If a record is being added to a random file, the PRU address is returned in the key area (the contents of the key area have no meaning when the call is made). If a record is added to a sequential file, there is no key, so the key descriptor and its associated information really describe the first element to be placed in the record.

The ADDR command must be used before any put type commands may be used and is called using one of the following statements.

COBOL	ENTER ADDR USING filename, data base name, status word, return flag, key descriptor, key area, element descriptor, element area, ..., element descriptor, element area.
FORTRAN Extended	CALL ADDR (filename, data base name, status word, return flag, key descriptor, key area, length, type,element descriptor, element area, length type, ..., element descriptor, element area, length, type)
COMPASS	The order of the parameters in the argument list is: filename, data base name, status word, return flag, key descriptor, key area, length, type, element descriptor, element area, length, type, ..., element descriptor, element area, length type.

BLKGET

This command results in the retrieval of a physically contiguous block of records from the data base. This block is placed in a data manager buffer and must not exceed the maximum number of PRUs allowed for this request. The number is an assembly constant (MAXBLK), normally set to 64, in deck COMBDBM. The request waits for the required buffer space to be available before beginning processing. The number of records, n, is specified in the command. The user must provide a buffer n*2 words in length that will receive the keys from the n contiguous records. These keys are returned by the data manager after the block is read into its buffer. Each key is left-justified and in the same format as it appears in the data base. Two central memory words are allowed for each key. At this point the user must use a key to obtain a record or element. If the user has a block of records from a file in memory from a prior BLKGET and makes a new BLKGET call or any keyed request of a record outside the block, the data manager automatically releases all records, except the last one, in the block. The command is not valid for sequential files. It is called by one of the following statements.

COBOL	ENTER BLKGET USING filename, data base name, status word, return flag, key descriptor, key value, number of records, key buffer area.
FORTRAN Extended	CALL BLKGET (filename, data base name, status word, return flag, key descriptor, key value, length, type, number of records, length, type, key buffer area, length, type)
COMPASS	The order of the parameters in the argument list is: filename, data base name, status word, return flag, key descriptor, key value, length, type, number of records, length, type, key buffer area, length, type.

BLKPUT

This command causes a contiguous block of records to be transferred from a data manager buffer to the data base. The number of records, n, is specified in the command. Prior to this command, a BLKGET command on a contiguous block containing this block of records must have been done. The number of records in the BLKPUT command must not exceed the number specified in the BLKGET command. The first record in the block to be written must be the same as the first record of the BLKGET.

If any type of an error is received by the user on a BLKPUT request, the user can recover by releasing all buffer space for the file. When the buffer space is released (by means of RELES or RELESAL), no data will be written on disk. The command is called by one of the following statements.

COBOL	ENTER BLKPUT USING filename, data base name, status word, return flag, key descriptor, key value, number of records.

FORTRAN Extended	CALL BLKPUT (filename, data base name, status word, return flag, key descriptor, key value, length, type, (number of records, length, type)
COMPASS	The order of the parameters in the argument list is: filename, data base name, status word, return flag, key descriptor, key value, length, type, number of records, length, type.

CLEAR

This command replaces all updated data base records and returns all user hold buffers, locks, and BSTs. The command is called by one of the following statements.

COBOL	ENTER CLEAR USING status word, return flag.
FORTRAN Extended	CALL CLEAR (status word, return flag)
COMPASS	The order of the parameters in the parameters list is: status word, return flag.

DMSTAT

This command causes the record-status-entries (RSE) and the number of RSEs found to be returned to a specified buffer area.

In addition, COBOL and FORTRAN Extended uers may specify which bit(s) is (are) to be examined with the set/not-set condition of said bit(s) in each RSE being returned to the corresponding status bit buffer(s) in unnormalized floating point format (refer to appendix A).

The first three parameters are required on all DMSTAT calls. Parameters beyond the status word parameter are allowed in COBOL and FORTRAN Extended calls only. COMPASS users can independently examine selected bits returned in the RSE buffer.

It is possible for two users to access the same record or file and each user has a separate BST.

This function is implemented with auto recall forced and therefore the return flag is not present as on most commands.

The command is called by one of the following statements.

COBOL	ENTER DMSTAT USING RSE buffer, number, status word, status bit, status bit buffer, ..., status bit, status bit buffer.
FORTRAN Extended	CALL DMSTAT (RSE buffer, number, status word, status bit, status bit buffer, ..., status bit, status bit buffer)
COMPASS	The order of the parameters in the argument list is: RSE buffer, number, status word.

GETB

This command retrieves element(s) from the preceding record in the file without locking the record. The previous record is determined by the type of chaining used in the file. The file must have been accessed or positioned by the user prior to this call.

NOTE

If the user holds more than one record from a file (that is, NREC is greater than 1) and issues a command that does not use a key, such as GETNR, then the last record into memory is used as the positioning record.

The command is called by one of the following statements.

COBOL	ENTER GETB USING filename, data base name, status word, return flag, element descriptor, element area, ..., element descriptor, element area.
FORTRAN Extended	CALL GETB (filename, data base name, status word, return flag, element descriptor, element area, length, type, ..., element descriptor, element area, length, type)
COMPASS	The order of the parameters in the argument list is: filename, data base name, status word, return flag, element descriptor, element area, length, type, ..., element descriptor, element area, length, area.

GETBL

This command performs the same function as GETB but locks the record retrieved. It is called by one of the following statement. Refer to note in GETB paragraph.

COBOL	ENTER GETBL USING filename, data base name, status word, return flag, element descriptor, element area, ..., element descriptor, element area.

FORTRAN Extended	CALL GETBL (filename, data base name, status word, return flag, element descriptor, element area, length, type, ..., element descriptor, element area, length, type)
COMPASS	The order of the parameters in the argument list is: filename, data base name, status word, return flag, element descriptor, element area, length, type, ..., element descriptor, element area, length, type.

GETN

This command retrieves element(s) from the next record in the data base without locking that record. Refer to note in GETB paragraph. The next record is determined by the type of chaining used in the file. The file must have been accessed or positioned by the user prior to this call. The command is called by one of the following statements.

COBOL	ENTER GETN USING filename, data base name, status word, return flag, element descriptor, element area, ..., element descriptor, element area.
FORTRAN Extended	CALL GETN (filename, data base name, status word, return flag, element descriptor, element area, length, type, ..., element descriptor, element area, length, type)
COMPASS	The order of the parameters in the argument list is: filename, data base name, status word, return flag, element descriptor, element area, length, type, ..., element descriptor, element area, length, type.

GETNL

This command performs the same function as GETN, but locks the record containing the element(s) requested. Refer to note in GETB paragraph. This command is called by one of the following statements.

COBOL	ENTER GETNL USING filename, data base name, status word, return flag, element descriptor, element area, ..., element descriptor, element area.
FORTRAN Extended	CALL GETNL (filename, data base name, status word, return flag, element descriptor, element area, length, type, ..., element descriptor, element area, length, type)
COMPASS	The order of the parameters in the argument list is: filename, data base name, status word, return flag, element descriptor, element area, length, type, ..., element descriptor, element area, length, type.

GETR

This command retrieves a record from the data base without locking that record. The record is placed in a predefined user area as well as in the unique buffer area for the file. Sequential files may not be accessed by this command. The command is called by one of the following statements.

COBOL	ENTER GETR USING filename, data base name, status word, return flag, key descriptor, key area, data area.
FORTRAN Extended	CALL GETR (filename, data base name, status word, return flag, key descriptor, key area, length, type, data area, length, type)
COMPASS	The order of the parameters in the argument list is: filename, data base name, status word, return flag, key descriptor, key area, length, type, data area, length, type.

GETRL

This command performs the same function as GETR; it also locks the record. The command is called by one of the following statements.

COBOL	ENTER GETRL USING filename, data base name, status word, return flag, key descriptor, key area, data area.
FORTRAN Extended	CALL GETRL (filename, data base name, status word, return flag, key descriptor, key area, length, type, data area, length, type)
COMPASS	The order of the parameters in the argument list is: filename, data base name, status word, return flag, key descriptor, key area, length, type, data area, length, type.

GETNR

This command retrieves the next record in the file without locking that record. The next record depends on the type of chaining used in the file. Refer to note in GETB paragraph. The file must have been previously accessed or positioned by the user. The command is called by one of the following statements.

COBOL	ENTER GETNR USING filename, data base name, status word, return flag, data area.
FORTRAN Extended	CALL GETNR (filename, data base name, status word, return flag, data area, length, type)
COMPASS	The order of the parameters in the argument list is: filename, data base name, status word, return flag, type, data area, length, type.

GETNRL

This command performs the same function as GETNR but also locks the record retrieved. Refer to note in GETB paragraph. The command is called by one of the following statements.

COBOL	ENTER GETNRL USING filename, data base name, status word, return flag, data area.
FORTRAN Extended	CALL GETNRL (filename, data base name, status word, return flag, data area, length, type)
COMPASS	The order of the parameters in the argument list is: filename, data base name, status word, return flag, type, data area, length, type.

GETRB

This command retrieves the preceding record in a file without locking that record. The preceding record depends on the type of chaining used in the file. Refer to note in GETB paragraph. The file must have been accessed or positioned by the user prior to this call. The command is called by one of the following statements.

COBOL	ENTER GETRB USING filename, data base name, status word, return flag, data area.
FORTRAN Extended	CALL GETRB (filename, data base name, status word, return flag, data area, length, type)
COMPASS	The order of the parameters in the argument list is: filename, data base name, status word, return flag, type, data area, length, type.

GETRBL

This command performs the same function as GETRB; it also locks the record retrieved. Refer to note in GETB paragraph. This command is called by one of the following statements.

COBOL	ENTER GETRBL USING filename, data base name, status word, return flag, data area.
FORTRAN Extended	CALL GETRBL (filename, data base name, status word, return flag, data area, length, type)
COMPASS	The order of the parameters in the argument list is: filename, data base name, status word, return flag, type, data area, length, type.

GETT

This command retrieves element(s) from the data base without locking the record containing the elements. The record is placed in the unique buffer for that file and the specified elements are transferred to the user. The command may not be used for sequential files and is called by one of the following statements.

COBOL	ENTER GETT USING filename, data base name, status word, return flag, key descriptor, key area, element descriptor, element area, ..., element descriptor, element area.
FORTRAN Extended	CALL GETT (filename, data base name, status word, return flag, key descriptor, key area, length, type, element descriptor, element area, length, type, ..., element descriptor, element area, length, type)
COMPASS	The order of the parameters in the argument list is: filename, data base name, status word, return flag, key descriptor, key area, length, type, element descriptor, element area, length, type, ..., element descriptor, element area, length, type.

GETL

This command retrieves elements from the data base and locks the record containing the elements requested. The command may not be used to access sequential files and is called by one of the following standards.

COBOL	ENTER GETL USING filename, data base name, status word, return flag, key descriptor, key area, element descriptor, element area, ..., element descriptor, element area.

FORTRAN Extended	CALL GETL (filename, data base name, status word, return flag, key descriptor, key area, length, type, element descriptor, element area, length, type, ..., element descriptor, element area, length, type)
COMPASS	The order of the parameters in the argument list is: filename, data base name, status word, return flag, key descriptor, key area, length, type, element descriptor, element area, length, type, ..., element descriptor, element area, length, type.

LOCKF

This command locks a specified file in a given data base, making the file inaccessible to other transactions attempting to update records in the file. It thus prevents one transaction from modifying a file while another transaction has it locked. The file may still be read by other transactions.

COBOL	ENTER LOCKF USING filename, data base name, status word, return flag.
FORTRAN Extended	CALL LOCKF (filename, data base name, status word, return flag)
COMPASS	The order of the parameters in the argument list is: filename, data base name, status word, return flag.

The file lock is released by the CEASE, CLEAR, RELESAL, and UNLOCKAL commands.

OFFTRCE

The OFFTRCE command turns off the tracing for a particular record of a traced file. When used, this command should be executed before writing the record into the data base (that is, before issuing a put type request). The command cannot be used for sequential files. It is called by one of the following statements.

COBOL	ENTER OFFTRCE USING filename, data base name, status word, return flag, key descriptor, key area.
FORTRAN Extended	CALL OFFTRCE (filename, data base name, status word, return flag, key descriptor, key area, length, type)
COMPASS	The order of the parameters in the argument list is: filename, data base name, status word, return flag, key descriptor, key area, length, type.

ONTRCE

The ONTRCE command truns on tracing for a particular record of a traced file. The command is used to rescind a previous OFFTRCE command amd is subject to the same constraints. The command is called by one of the following statements.

COBOL	ENTER ONTRCE USING filename, data base name, status word, return flag, key descriptor, key area.
FORTRAN Extended	CALL ONTRCE (filename, data base name, status word, return flag, key descriptor, key area, length, type)
COMPASS	The order of the parameters in the argument list is: filename, data base name, status word, return flag, key descriptor, key area, length, type.

PURGER

This command sets the purge bit in a specific record in the data base, indicating that the record is inactive, and zero fills the record (that is, all data is lost). If the file is simple-chained, the chain pointers remain in the record. If the file is complex-chained, the chain pointers are cleared and the record is removed from the chain. The purge bit is bit 59 of the first word of the record. The record may not be retrieved until after an ADDR is done. Both prime and dual copies are updated if the file is dual-recorded. Since a key is required, this command may not be used for sequentil files. The command is called by one of the following statments.

COBOL	ENTER PURGER USING filename, data base name, status word, return flag, key descriptor, key area.
FORTRAN Extended	CALL PURGER (filename, data base name, status word, return flag, key descriptor, key area, length, type)
COMPASS	The order of the parameters in the argument list is: filename, data base name, status word, return flag, key descriptor, key area, length, type.

PUT

This command inserts an element(s) into a record in the file's buffer. If the record does not exist in the file's buffer area, it is retrieved by the data manager and locked in a buffer. The command cannot be used for sequential files and is called by one of the following statements.

COBOL	ENTER PUT USING filename, data base name, status word, return flag, key descriptor, key area, element descriptor, element area, ..., element descriptor, element area.

FORTRAN Extended	CALL PUT (filename, data base name, status word, return flag, key descriptor, key area, length, type, element descriptor, element area, length, type, ..., element descriptor, element area, length type)
COMPASS	The order of the parameters in the argument list is: filename, data base name, status word, return flag, key descriptor, key area, length, type, element descriptor, element area, length type, ..., element descriptor, element area, length, type.

PUTF

This command performs the same function as PUT, but it forces the modified record to be written into the data base immediately. The buffer space allocated for this file wil be released. This command cannot be used for sequential files and is called by one of the following statements.

COBOL	ENTER PUTF USING filename, data base name, status word, return flag, key descriptor, key area, element descriptor, element area, ..., element descriptor, element area.
FORTRAN Extended	CALL PUTF (filename, data base name, status word, return flag, key descriptor, key area, length, type, element descriptor, element area, length, type, ..., element descriptor, element area, length type)
COMPASS	The order of the parameters in the argument list is: filename, data base name, status word, return flag, key descriptor, key area, length, type, element descriptor, element area, length type, ..., element descriptor, element area, length, type.

PUTI

This command inserts an element(s) into the record currently in the file's buffer area. A key is not needed with this type of PUT, but a file access must have previously placed the record in the buffer. This type of PUT command is especially useful for sequential files. The command is called by one of the following statements.

COBOL	ENTER PUTI USING filename, data base name, status word, return flag, element descriptor, element area, ..., element descriptor, element area.
FORTRAN Extended	CALL PUTI (filename, data base name, status word, return flag, element descriptor, element area, length, type, ..., element descriptor, element area, length type)
COMPASS	The order of the parameters in the argument list is: filename, data base name, status word, return flag, element descriptor, element area, length type, ..., element descriptor, element area, length, type.

PUTIF

This command performs the same function as PUTI, but it forces the updated record to be written immediately. PUTIF is called by one of the following statements.

COBOL	ENTER PUTIF USING filename, data base name, status word, return flag, element descriptor, element area, ..., element descriptor, element area.
FORTRAN Extended	CALL PUTIF (filename, data base name, status word, return flag, element descriptor, element area, length, type, ..., element descriptor, element area, length type)
COMPASS	The order of the parameters in the argument list is: filename, data base name, status word, return flag, element descriptor, element area, length type, ..., element descriptor, element area, length, type.

PUTR

This command replaces a record in the file's buffer area and leaves the record in a locked condition. If the record does not exist in the file's buffer area, it is retrieved from disk and placed in a locked buffer. On all PUT record commands, the data manager saves the key value of the record in the buffer area before the record is trnsferred from the user area and then replaces this value after the transfer. If the user changes the key, the change is ignored by the data manager. The command may not be used for sequential files and is called by one of the following statements.

COBOL	ENTER PUTR USING filename, data base name, status word, return flag, key descriptor, key area, data area.
FORTRAN Extended	CALL PUTR (filename, data base name, status word, return flag, key descriptor, key area, length, type, data area, length, type)

COMPASS	The order of the parameters in the argument list is: filename, data base name, status word, return flag, key descriptor, key area, length, type, data area, length, type.

PUTRF

This command performs the same function as PUTR, but it forces the record to be written into the data base immediately. The command may not be used for sequential files and is called by one of the following statements.

COBOL	ENTER PUTRF USING filename, data base name, status word, return flag, key descriptor, key area, data area.
FORTRAN Extended	CALL PUTRF (filename, data base name, status word, return flag, key descriptor, key area, length, type, data area, length, type)
COMPASS	The order of the parameters in the argument list is: filename, data base name, status word, return flag, key descriptor, key area, length, type, data area, length, type.

PUTRI

This command inserts a record into the file's buffer area replacing the record currently held. A key is not needed with this type of PUT, but the file must have been accessed by the user prior to this call. This command is especially useful for sequential files. The PUTRI command is called by one of the following statements.

COBOL	ENTER PUTRI USING filename, data base name, status word, return flag, data area.
FORTRAN Extended	CALL PUTRI (filename, data base name, status word, return flag, data area, length, type)
COMPASS	The order of the parameters in the argument list is: filename, data base name, status word, return flag, data area, length, type.

PUTRIF

This command performs the same function as PUTRI, but it forces the record to be written to the data base immediately. The PUTRIF command is called by one of the following statement.

COBOL	ENTER PUTRIF USING filename, data base name, status word, return flag, data area.
FORTRAN Extended	CALL PUTRIF (filename, data base name, status word, return flag, data area, length, type)
COMPASS	The order of the parameters in the argument list is: filename, data base name, status word, return flag, data area, length, type.

READEDT

This command returns element descriptors for a specified data base and file from the EDT; it also returns the total number of descriptors for the file in the EDT and the number of descriptors returned by this command.

NOTE

Check with installation personnel before using this command. Each site can determine whether or not this command is available for use.

When this command is used in nonbatch mode by a task under TAF control, it can access only those element descriptors on the data base or bases which the task is allowed to access. This is determined by the data base associated with the user number or terminal name with which the user entered the transaction subsystem.

All six parameters are required on all READEDT calls. Although the form of a COBOL call is given, it is not recommended that this COBOL form be used because it is difficult to extract the information from the fields of the element descriptors using COBOL. It is recommended that a COBOL user call a COMPASS or FORTRAN subroutine to obtain and unpack the element descriptors. The READEDT command is called by one of the following statements.

COBOL	ENTER READEDT USING filename, data base name, status word, return flag, EDT address, EDT buffer size.
FORTRAN Extended	CALL READEDT (filename, data base name, status word, return flag, EDT address, EDT buffer size)
COMPASS	The order of the parameters in the argument list is: filename, data base name, status word, return flag, EDT address, EDT buffer size.

REPOS

This command repositions the record pointer. There are two general types of reposition available.

- Reposition entire file
- Reposition within a chain of specified file

Both types are positioned to the beginning or to the end, depending on the count parameter. If count 0, the file or chain is rewound. If count ≥ 0, the file or chain is positioned to the last record.

To reposition by chain, the file must be defined through DATADEF as a complex-chained file.

To reposition the entire file, the chain descriptor and chain area parameters must be present, as they must be when repositioning within a chain. However, when repositioning the entire file, or when repositioning a simple-chained file, the chain descriptor and chain area parameters must be equal to zero. If a reposition command rewind the file, a GETN command will cause the data manager to read the first record in the file. REPOS is called by one of the following statements.

COBOL	ENTER REPOS USING filename, data base name, status word, return flag, chain descriptor, chain area, count.
FORTRAN Extended	CALL REPOS (filename, data base name, status word, return flag, chain descriptor, chain area, length, type, count, length, type)
COMPASS	The order of the parameters in the argument list is: filename, data base name, status word, return flag, chain descriptor, chain area, length, type, count, length, type.

RECHAIN

This command moves a record from a given chain in a complex-chained file to another defined chain within the same file. Elements may also be added to the record at the same time. Both prime and dual copies are updated if the file is dual recorded. The command is called by one of the following statement.

COBOL	ENTER RECHAIN USING filename, data base name, status word, return flag, key descriptor, key area 1, key descriptor, key area 2, element descriptor, element area, ..., element descriptor, element area.
FORTRAN Extended	CALL RECHAIN (filename, data base name, status word, return flag, key descriptor, key area 1, length, type, key descriptor, key area 2, length, type, element desriptor, element area, length, type, ..., element descriptor, element area, length, type)
COMPASS	The order of the parameters in the argument list is: filename, data base name, status word, return flag, key descriptor, key area 1, length, type, key descriptor, key area 2, length, type, element descriptor, element area, length, type, ..., element descriptor, element area, length, type.

key area 1 — The record pointer (key of record in chained file)

key area 2 — The new chain name (primary key of new chaining record in chainer file)

filename — Name of the chained file

RELES

This command releases buffer space allocated for the record specified unless the user is in block mode. If using BLKGET or BLKPUT, this operation is ignored because any subsequent access outside the block causes the entire block to be released. The command may not be used for sequential files and is called by one of the following statements.

COBOL	ENTER RELES USING filename, data base name, status word, return flag, key descriptor, key area.
FORTRAN Extended	CALL RELES (filename, data base name, status word, return flag, key descriptor, key area, length, type)
COMPASS	The order of the parameters in the argument list is: filename, data base name, status word, return flag, key descriptor, key area, length, type.

RELESAL

This command releases all buffer space allocated to the user. No records are written on disk. The command is called by one of the following statements.

COBOL	ENTER RELESAL USING status word, return flag.
FORTRAN Extended	CALL RELESAL (status word, return flag)
COMPASS	The parameters in the argument list are status word and return flag.

The command must be used after BLKGET command if PURGE or ADDR commands are to be performed later in the task.

STATCHK

The STATCHK command is designed to interrogate the status word returned by the data manager. The appropriate bits (0, 6 through 14, 15 through 22) are checked and the errors are returned to the user in display code.

STATCHK may be called by FORTRAN or COBOL programs with the following format:

COBOL	ENTER STATCHK USING status-word, message-field, non-fatal-error.
FORTRAN Extended	CALL STATCHK (status-word, message-field, non-fatal-error)
Status-word	Is the same status word used by a data manager command.
message field	Data area in user program in which the error code is to be returned (tables 6-2 and 6-3). This should be in right justified integer format.
non-fatal-error	Non-fatal error data area in user program.

In COBOL, status-word should be computational-1, with message-field and non-fatal-error both being display code (9(3)).

The status-word and message-field parameters are required with non-fatal-error being optional. Table 6-2 lists the responses entered in message-field and their meanings. The non-fatal-error field is used mainly with dual-recorded files. Table 6-3 lists the responses entered in non-fatal-error and their meanings.

UNLOK

This command unlocks a specified record in a given file and data base; however, it may not be used for sequential files. If the specified record has been modified, it is written into the data base. The command is called by one of the following statements.

COBOL	ENTER UNLOK USING filename, data base, name, status word, return flag, key descriptor, key area.
FORTRAN Extended	CALL UNLOK (filename, data base name, status word, return flag, key descriptor, key area, length, type)
COMPASS	The order of the parameters in the argument list is: filename, data base name, status word, return flag, key descriptor, key area, length, type.

UNLOKAL

This command unlocks all previously locked records, which were locked by tasks in its task chain. All the updated records are replaced in the data base. The command is called by one of the following statements.

COBOL	ENTER UNLOKAL USING status word, return flag.
FORTRAN Extended	CALL UNLOKAL (status word, return flag)
COMPASS	The parameters in the argument list are status word, and return flag.

UNLOKF

This command unlocks a previously locked file, thus making it available to all transactions. The command is called by one of the following statement.

COBOL	ENTER UNLOKF USING filename, data base name, status word, return flag.
FORTRAN Extended	CALL UNLOKF (filename, data base name, status word, return flag)
COMPASS	The parameters in the argument list is: filename, data base name, status word, return flag.

PACKED FORMAT CALLING SEQUENCE

The primary advantages of the packed format calling sequence are the capability to process any number of elements in one data manager call and a significant reduction in the amount of memory used on the calls.

A packed format calling sequence is available for the following element-processing functions.

ADDR	GETL	GETT	PUTI
GETB	GETN	PUT	PUTIF
GETBL	GETNL	PUTF	RECHAIN

The calling format is:

COBOL	ENTER func USING filename, data base name, status word, return flag, key descriptor, key area, fwa.
FORTRAN Extended	CALL func (filename, data base name, status word, return flag, key descriptor, key area length, type, fwa)
COMPASS	The order of the parameters in the argument list is: filename, data base name, status word, return flag, key descriptor, key area, length, type, fwa, zero word.
fwa	First word address of the packed parameter(s).
func	Name of the particular function requested.

TABLE 6-2. PREVENTIVE ERRORS - MESSAGE FIELD

Code	Description
000	Successful data manager request
001	Disk surface error(principal record)
002	Disk logic error (principal record)
003	Disk surface error (other record)
004	Disk logic error (other record)
005	Disk logic error writing record from previous function
006	Illegal access method requested
010	Invalid file name
011	Invalid data base name
012	Invalid key descriptor
013	Key descriptor is not key type element
015	Invalid contents of key area
016	Key converts to record address of zero (records must start with value 1 or larger)
017	Key hashes to nonexistent disk address
020	Group element contains a group element
021	Group element contains no elements
022	User or EDT length invalid
023	Element illegally crosses word boundary
024	Illegal termination of user parameter list
030	No record currently held (GETN or GETB)
031	Purged record requested
032	Key value already assigned; possible attempt to do an ADDR on a record that already contains data
033	File not complex-chained (RECHAIN)
034	Invalid count (REPOS)
035	No data area given (GET or PUT record)
036	Security violation
037	No record held on UNLOK or RELES, or no file held on UNLOCKF, or no record in memory on ONTRCE or OFFTRCE
040	Attempt to purge already purged record
042	User buffer length less than record size
043	End-of-chain encountered on GETN or GETB type of request. Equivalent to EOF function in FORTRAN or AT END in COBOL
044	Area not large enough to hold all key values in block
045	Error in number of records parameter
046	BLKPUT number of records exceeds number of records held
047	Unable to do block write because of hardware problems
050	User's data overflows data base field
051	Attempt to convert 9-type field that is greater than 18 characters
052	Invalid/missing element type or bad parameter address
053	Invalid data type code
054	Invalid conversion request; key area data type does not match key descriptor data type
055	Nonexistent element requested
056	Integer too large for 10-character alpha conversion
057	Number of records exceeds allowable buffer size
060	Block request address goes beyond end-of-information
061	Chainer element not specified
062	Chainer record not active or empty if ADDR is being attempted
063	Chained record already in specified chain (RECHAIN)
064	Nonkeyed request made when in block mode
065	Cannot do block read with pooled record contained within block
066	Illegal write on read only file
067	Index file not attached
070	User address out of range
071	User buffer area extends beyond field length (DMSTAT or READEDT) or error in buffer size (READEDT)
072	Required parameter missing (READEDT)
076	System error because of invalid function code. This error will occur upon use of READEDT when use is disabled by the site (which is the default)
077	System error because of faulty EDT
775	Status bit buffer missing (DMSTAT)
776	Illegal status bit specified in DMSTAT call
777	Data manager command processing was not complete when STATCHK processed the status word; in other words, the return flag was not set to value of 1 in an ADDR, GETx or PUTx

TABLE 6-3. NONPREVENTIVE ERRORS - NON-FATAL-ERROR FIELD

Code	Meaning
xx xxx xx1	Surface error, principal record, first recording of dual-recorded file
xx xxx x1x	Surface error, principal record, second recording of dual-recorded file
xx xxx 1xx	Logic error, principal record, first recording of dual-recorded file
xxx xx1 xxx	Logic error, principal record, second recording of dual-recorded file
xx x1x xxx	Surface error, other record, first recording of dual-recorded file
xx 1xx xxx	Surface error, other record, second recording of dual-recorded file
x1 xxx xxx	Logic error, other record, first recording of dual-recorded file
1x xxx xxx	Logic error, other record, second recording of dual-recorded file

The parameters relating to the key are not present in the GETB, GETBL, GETN, GETNL, PUTI, and PUTIF calls. In the ADDR and RECHAIN calls the chain descriptor, chain area, length, and type† parameters must appear after the key type and before the first packed parameter address.

The packed parameter is formed using the COMPASS pseudo instruction VFD in the following format.

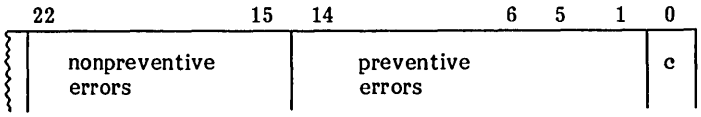
VFD	18/eld,6/type,18/len,18/addr
eld	Element descriptor
type	Format of the data as it exists in the user's storage area (U, F, 9, and so forth)
len	Length of the data in characters.
addr	Address of element area

The last packed parameter must be followed by a zero word. Figure 6-1 illustrates the use of a normal call versus a packed format call. Since this is a batch program, the common block specified as BB is not the communication block. The bracketed calls in the figure produce the same results.

†Length and type are not usually present in COBOL calls.

ERROR PROCESSING

Upon completion of any data manager request, a word (as described below) is returned to the user. The word must be set to all zeros (binary) prior to any data manager call.



c Completion bit

Status Word	Definition
preventive error	Error that prevents completion of the data manager request. These are returned in the message-field (second parameter) of the STATCHK command (table 6-2).
nonpreventive error	Hardware errors encountered while accessing copy 1 of a dual-recorded file do not prevent completion of data manager requests. These are entered in the non-fatal-error field (third parameter) of the STATCHK command (table 6-3).

The following terms are used in the status word tables which explain the types of nonpreventive and preventive errors reported.

surface error	Data stored on disk surface is in error.
logic error	Hardware error (except surface error) detected during input/output.
principal record	Actual record called for. On multiple file operations like RECHAIN, principal records are distinguished from other records.
other record	Records other than the principal record which must be referenced or modified during multiple file operations like RECHAIN.

The following situations may be reflected in the settings of the status word.

Completion bit	Set when data manager request is complete.
Nonpreventive errors	Bits 15 through 22 (these apply only to dual-recorded files when executing a read either copy or a write both copy).

```
      PROGRAM ZCR (Z3,TAPE3=Z3,OUTPUT)
      COMMON /BB/B(70)
      EQUIVALENCE (C,B(61)), (FL,B(63))


          .
          .    initialize constants and set up arrays
          .

      DATA (F1=4LSONA), (F2=4LSOWA), (F3=4LSNNA), (F4=4LSNWA)
      DATA (A=1LA), (D=1LD), (F=1LF), (I=1LI), (L=1LL), (R=1LR),

      DATA (U=1LU), (X=1LX), (ZZ=2LZZ)

          .
          .    set up additional constants
          .

      CALL ADDR(F1,ZZ,F1A(M),P,3LACN,B(1),2,U)

      CALL PUTF(F1,ZZ,F1P1(M),P,3LACN,B(1),2,U,3LLNM,B(2),10,X,
     1 3LFNM,B(3), 10,X,3LDAT,B(4), 10,X,3LMT,B(5), 4, U,
     2 3LRAT,B(6), 10,R,3LMS1,B(7), 10,X,3LMS2,B(8), 10, X,
     3 3LMS3,B(9), 30,X,3LNPY,B(12), 2,U,3LAMO,B(13) 4, U,
     4 3LITD,B(14), 3,U,3LPRE,B(15),  3,U,3LLDT,B(16), 10, X,

      CALL PUTF(F1,ZZ,F1P2(M),P,3LACN,B(1),2,U,
     5 3LGRP,B(17,10,I,3LNIN,B(64),10,1L9,3LTEN,B(66),18,1L9,
     6 3LLOG,B(68),1,L,3LDBL,B(61),20,D,3LFLT,B(63),10,F)

      .
      .    other processing
      .
      CALL ADDR(F3,ZZ,F3A(M),P,3LACN,B(1),2,U,B(21))

      .
      .    other processing
      .
      END

               IDENT CALL
               USE /BB/
      D        BSSZ        20
               VFD 18/3LLNM,6/1LX,18/10,18/D+1
               VFD 18/3LFNM,6/1LX,18/10,18/D+2
               VFD 18/3LDAT,6/1LX,18/10,18/D+3
               VFD 18/3LAMT,6/1LU,18/04,18/D+4
               VFD 18/3LRAT,6/1LR,18/10,18/D+5
               VFD 18/3LMS1,6/1LX,18/10,18/D+6
               VFD 18/3LMS2,6/1LX,18/10,18/D+7
               VFD 18/3LMS3,6/1LX,18/30,18/D+8
               VFD 18/3LNPY,6/1LU,18/02,18/D+11
               VFD 18/3LAMO,6/1LU,18/04,18/D+12
               VFD 18/3LTTD,6/1LU,18/03,18/D+13
               VFD 18/3LPRE,6/1LU,18/03,18/D+14
               VFD 18/3LLDT,6/1LX,18/10,18/D+15
               VFD 18/3LGRP,6/1LI,18/10,18/D+16
               VFD 18/3LNIN,6/1L9,18/10,18/D+63
               VFD 18/3LTEN,6/1L9,18/18,18/D+65
               VFD 18/3LLOG,6/1LL,18/01,18/D+67
               VFD 18/3LDBL,6/1LD,18/20,17/D+60
               VFD 18/3LFLT,6/1LF,18/10,18/D+62
               BSSZ        1
               BSSZ        10
               USE *
               END
```

Figure 6-1. Normal Call Versus a Packed Format Call

BATCH ACCESS TO THE DATA MANAGER

All of the data manager commands described previously are available for use by batch programs as well as by tasks. A program called BDMI provides the batch interface to the data manager.

BDMI does not attach any files. Therefore, the user must attach all files, including the EDT file, in the correct mode to allow performance of the desired functions. For example, if he wishes to alter a file, he must attach it in write or modify mode. If transaction faciity is running, the mode of attachment must be compatible with the mode in which transaction facility has attached the file. Refer to section 3, Attach-Mode.

The batch user has a read and update security of 6.

Three different types of requests are processed by BDMI. They are:

CTI	Call transaction facility interface
DBA	Data base access
SCT	Schedule tasks

Refer to Summary of Commands later in this section for specific requests included in each category.

Upon receipt of the first CTI, SCT, or DBA type of request, BDMI examines the control statement to extract the data base name and to determine if the trace option has been selected (refer to Trace Option later in this section). If the trace option is selected, BDMI writes a coded image of the call and its parameters to the trace file. CTI nd SCT requests are used to make requests to the transaction facility. No execution of these requests is done by BDMI except for the CEASE request. The CEASE request causes the data manager to write out all records on which the user has issued PUT type requests, but does not cause program termination. In a batch program that is being traced, the last request must be a CEASE to ensure that all information gets written onto the trace file.

On receiving the first DBA type of request, BDMI requests additional memory for EDT and buffer storage. The EDT is loaded to central memory from the data base description file, and all files specified in the EDT are checked to determine if they have been attached. If any file is not attached, its name is removed from the EDT, causing any data manager requests for that file to be rejected. DBA requests are then passed to the data manager for execution. Data base tracing (distinct from request tracing as described under Trace Option) occurs as in the transaction environment, except that trace files must be defined and attached by the user.

TRACE OPTION

The trace option can be a valuable aid in debugging tasks, since it provides a trace on all CTI, SCT, and DBA requests. The option is specified on the control statement that initiates the execution of the batch program. The data base name must also be specified on the control statement. Information contained in the trace entry includes the time of the call, the calling address, and an image of the request and all associated parameter values. More information is available later in this section. The user must not trace to a file that could be active, such as the file OUTPUT.

The following are legal forms of the control statement to execute a batch program.

LGOB.xx

LGOB.xx.

LGOB.xx,*

LGOB.xx,*filename

LGOB.xx,*filename

LGOB	Binary program file
xx	Data base name
*	Selects trace option
filename	Name of file on which trace records are written; default is TRACE

The format of the output for each track request is shown in figure 6-2.

SUMMARY OF COMMANDS

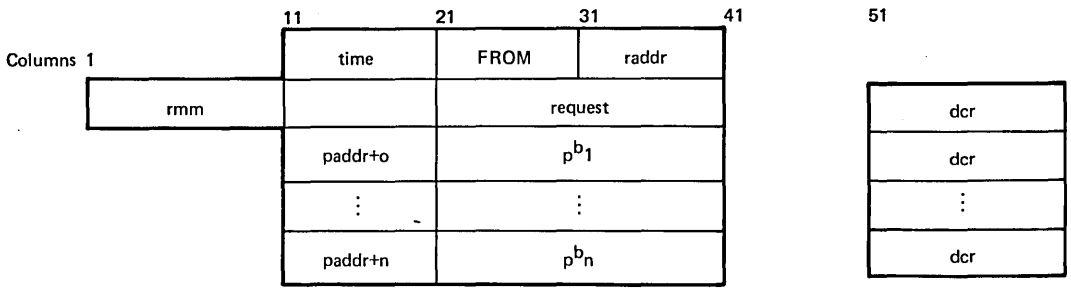
The following is a list of all data manager type requests. They are all executed fully in BDMI and are callable through COBOL, COMPASS, and FORTRAN. A trace is made of all requests if the option is selected.

ADDR	PURGER
BLKGET	PUT
BLKPUT	PUTF
GETB	PUTI
GETBL	PUTIF
GETL	PUTR
GETN	PUTRF
GETNL	PUTRI
GETNR	PUTRIF
GETNRL	READEDT†
GETR	RECALAL
GETRB	RECHAIN
GETRBL	RELES
GETRL	RELESAL
GETT	REPOS
LOCKF	UNLOK
OFFTRCE	UNLOKAL
ONTRCE	UNLOKF

Table 6-4 lists the CTI and SCT requests and indicates the language through which they may be called. Except for the CEASE request, no processing occurs for these requests in BDMI. A trace of each request is recorded if specified. The SEND and JOURNL requests also trace the text of their respective calls.

Table 6-5 lists the differences in operation between programs originating from terminal transactions and those originating from batch.

†This command is available only if an installation option is enabled (the default is disabled).



time Hour, minute and second that request was received (ahh.mm.ss.).

raddr Left justified address from which the request was made.

rmn Request mnemonic consisting of four alphabetic characters.
Most CTI and SCT request mnemonics are listed in table 6-4.†
For DBA requests, the mnemonic is DBAC.
An argument error in a user interface routine could result in a mnemonic of ABRT.

requet Octal representation of the system request being traced.

dcr Display code representation of word in columns 21-40.

paddr Right justified address of the parameter or data block associated with the request. This field may be empty if the request has no parameter.

pb Contents in octal format of the word at the location referenced by paddr.

Figure 6-2. Trace Request Format

†Some reserved or internal functions that produce a trace are not described here. If you receive a trace of one of these functions and need to have it interpreted, consult a site analyst.

TABLE 6-4. CTI AND SCT REQUESTS

Command	Type		Request	Usable In		
	CTI	SCT	Mnemonic	Cobol	Compass	Fortran
BEGIN	X		BEGN	Y	N	Y
BWAITINP		X	BWAI	N	Y	N
CALLRTN		X	CRTN	Y	Y	Y
CALLTSK		X	CALC	Y	Y	Y
		X	CALT	Y	Y	Y
CEASE		X	CEAS	Y	Y	Y
CIO	X		CIIO	Y	Y	Y
CMDUMP	X		CDMP	Y	Y	Y
DSDUMP	X		DDMP	Y	Y	Y
GETABH	X		GABH	Y	Y	Y
ITL	X		CITL	Y	Y	Y
JOURNL	X		JOUR	Y	Y	Y
KPOINT	X		KPNT	Y	Y	Y
LOADCB	X		LDCB	Y	Y	Y
LOGT	X		LOGT	N	Y	N
NEWTRAN		X	NTRN	N	Y	N
RELSCB	X		RLCB	Y	Y	Y
SEND	X		SEND	Y	Y	Y
SETCHT	X		SCHT	Y	Y	Y
SUBMT	X		SUBM	Y	Y	Y
TARO	X		TARO	Y	Y	Y
TERMDEF	X		TDEF	Y	Y	Y
TPSTAT	X		TPST	N	Y	N
TRANCHK	X		TRNK	N	Y	N
TSIM	X		TSIM	Y	Y	Y
WAIT		X	WAIT	Y	Y	Y
WAITINP		X	WTIN	Y	Y	Y

TABLE 6-5. OPERATING DIFFERENCES

Feature	From Terminal	From Batch
Languages available	COBOL COMPASS, FORTRAN	COBOL, COMPASS FORTRAN
Runs under	TAF	BDMI
Externals satisfied from	TRANLIB	BDMLIB
Program (task) structure	Overlay	Not overlay
Communication block	Yes	No
DBA requests	Processed	Processed†
CTI and SCT requests	Processed	Not processed†,††

†A trace of these commands may be selected.

††The CEASE request is an exception. It causes data manager records with the put flag set to be written to mass storage but does not cause program termination.

The buffer status table (BST) is used by the data manager to manage all data manager record buffers. The record buffers and BSTs are created by tasks issuing data manager commands.

Although the DMSTAT command allows the user to access all of the BST status bits, only three are of value to a user. These are:

Bit	Meaning of Set Condition
0	The lock bit. This record is locked by this transaction.
1	The put bit. A put command has been issued on this record. It has not yet been written to disk.
8	The block bit. This record is part of a block request.

The other 9 bits are used primarily by the system I/O routines and are not explained in this manual.

The following is an example of a COBOL call to DMSTAT and results returned by DMSTAT. This program illustrates the retrieval of the lock and put status bits for up to 10 records in core assigned to this user.

```
Identification Division.
    .
    .
    .
Working Storage Section.

77  MNUM      PIC 9(10)    COMP-1    VALUE 10.
77  LOCKS     PIC 9(10)    COMP-1    VALUE 0.
77  PUTS      PIC 9(10)    COMP-1    VALUE 1.
77  STAT      PIC 9(10)    COMP-1    VALUE 0.
    .
    .
    .
01  BUF.
    02 NUM      PIC 9(10)    COMP-1    VALUE 0.
    02 FIL-KEY  PIC X(10)    OCCURS 30 TIMES.

01  LOCKBUF.
    02 LCKSTAT PIC 9(10) COMP-1 OCCURS 10-TIMES

01  PUTBUF.
    02 PUTSTAT PIC 9(10) COMP-1 OCCURS 10-TIMES.

    .
    .
    .
Procedure Division.

    .
    .
    .
    ENTER DMSTAT USING BUF MNUM STAT PUTS
    PUTBUF LOCKS LOCKFBUF.

    .
    .
    .
```

It is assumed that there were three files with one record each in memory at the time of the DMSTAT request. Accordingly, the DMSTAT function would return the information shown in figure A-1.

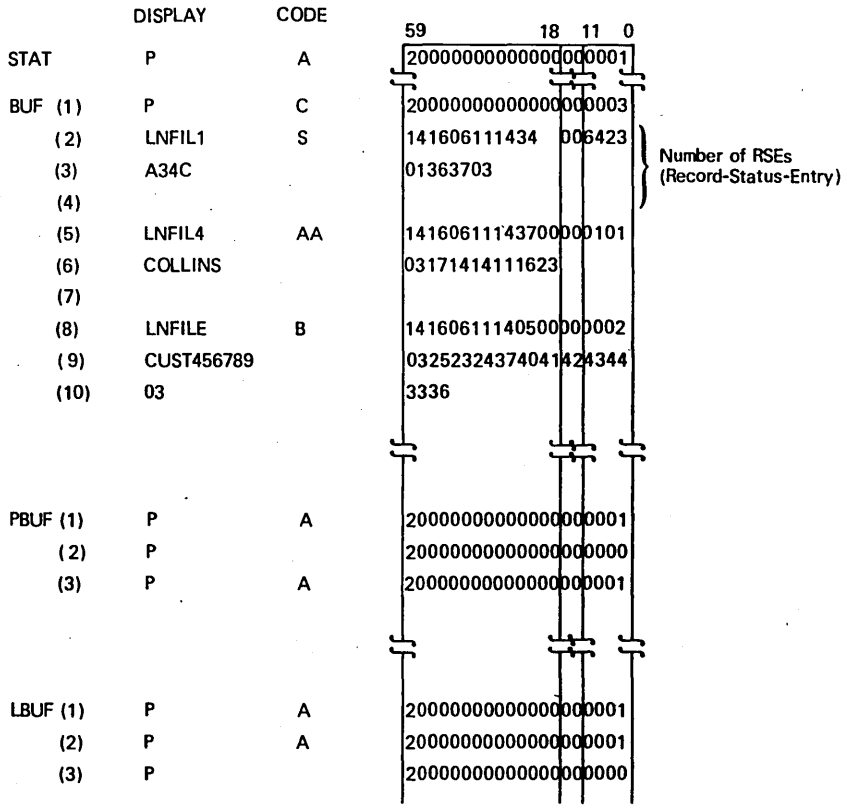


Figure A-1. DMSTAT Return

This appendix briefly describes the processing performed by the data manager for the combinations of element types possible.

Conversion Type	Entry Condition	Description of Processing
A. 9 to 9	1. The to field > from field.	Leading zeros padded for the difference.
	2. The to field < from field.	Excess digits in upper part of from field stripped off so that the sizes of the to and from fields are the same.
	3. From field > 18 digits.	Exit on error code 51.
B. 9 to X	1. Type 9 from field.	Leading zeros are stripped off and the length of from field redefined.
	2. The to field > from field.	Trailing blanks are padded for the difference.
	3. The to field < from field.	This occurs after operation B.1. Therefore, an overflow has occurred. The excess digits are stripped off as in A.2. and an overflow character (*) is added.
	4. Value of from field = binary zero.	Zero 9- code (type 0) are generated to fill to field.
	5. Value of from field-display zero.	This occurs after operation B.4. Trailing spaces are added after leftmost zero character in to field.
	6. Same as A.3.	Same as A.3.
	7. If from field is negative.	A leading (-) sign is added to the to field.
C. X to X	1. Same as B.2.	Same as B.2.
	2. Same as A.2.	Same as A.2.
	3. If from field-binary zero.	The to field is filled with blanks.
D. I,U to I,U	1. If from field > to field.	Excess is masked off upper part of from field.
	2. Type I-type I and from field < to field.	Upper bit of from field is sign extended in to field.
	3. Type I-type U.	Absolute value of from field is transferred.
E. L to I,U	1. From field is logically true.	Set to field to 1.
	2. From field is logically false.	Set to field to 0.
L to R,F	3. Same as E.1, E.2.	Same as E.1, E.2, except floating point representation.
F. I,U to L	1. If from field is = 0.	Set to field to 0 (logically false).
	2. If from field is $\neq 0$.	Set to field to -1 (logically true).
G. R,F to L	1. Same as F.1, F.2.	Same as F.1, F.2, except floating point representation.
H. R,F to I		Normalize, unpack, and scale down integer.
I. I,U to X	1. The from field > $10^{10}-1$.	Exit on error code 57.
	2. The from field < $10^{9}+1$.	Exit on error code 57.
J. R,F to X	1. Same as I.1, I.2.	Same as I.1, I.2.

All user-defined access methods are subject to the following constraints.

- Contents of registers B1 and B5 must not be destroyed
- The following parameters are passed to the subroutine in the registers specified

  (B4) First word address of contiguous memory that contains the key value

  (B5) Return address

  (X4) Bias value, right-justified

- The access method subroutine passes back the logical record number, right-justified, in X5
- All access methods return control to the calling program by executing a jump to the address specified by (B5)

Access method subroutines must be inserted into common deck COMBACM. An entry must be inserted in table TOAM, which is in deck COMBACM. This entry points to the new access method subroutine. The upper 42 bits contain the file name with which the access method is associated, and the lower 18 bits define the entry point for the associated access method subroutine. The ordinal or position in the table is the value for xx that is used when specifying ACCESS-METHOD IS xx in section 3, File-Description Section.

For example, assume a file QQQQ (in data base AA) is to be described in DATADEF that requires a new access method. The access method routine, with entry point AMN, is inserted into common deck COMBACM and an entry is made in the TOAM table. Assuming this is the third access method in COMBACM, the entry appears as the third word in TOAM in the following format.

```
VFD     42/AAQQQQ,18/AMN
```

Since it is the third entry in TOAM, the ordinal 3 is used in the ACCESS-METHOD clause in the file description of DATADEF.

When COMBACM is altered by the addition, deletion, or modification of an access method, the transaction executive, BDMI, and DBFORM must all be reassembled.

For example:

```
Bias value    -200,0008
Key value     353333343536 (in display code)
```

The access method would perform the following steps.

1. Convert from display code to integer

   $353333343536 \quad 200123_8$

2. Bias this amount

   $200123_8 - 200000_8 = 123_8$.

Thus, the logical record number equals 123_8.

This appendix contains an alphabetical listing of messages which could be of importance to the programmer using the TAF data manager. Each message is followed by an explanation of the message and/or the circumstances causing it to be issued, the recommended action, and the routine which issued the message.

Lowercase letters are used within a message to identify variable fields. All messages beginning with lowercase (variable) fields are listed alphabetically according to the first nonvariable field following the messages.

If you encounter a diagnostic or informative message that does not appear in this appendix, consult the NOS Diagnostic Index. This publication catalogs all messages produced by NOS and its products and specifies the manual or manuals in which each message is fully documented.

MESSAGE	SIGNIFICANCE	ACTION	ROUTINE
A NEW EDT OR INPUT DIRECTIVES REQUIRED.	Action must be specified by input directives or a new EDT indicated by the N parameter on the control statement.	Correct error and rerun.	DBFORM
ACCESS-METHOD CLAUSE REQUIRES PREVIOUS *PRIMARY-KEY* CLAUSE.	Self-explanatory.	Correct error and rerun.	DATADEF
ACCESS METHOD MUST BE .LE. 63.	Access methods must be numbered 1 through 63.	Correct error and rerun.	DATADEF
ATTEMPT TO DUPLICATE ACTION ON SAME FILE - filenam.	In one run, a user may not define a file more than once, copy it more than once, and so on.	Correct error and rerun.	DBFORM
BEGAN REFORMAT - filenam hh.mm.ss.	DBFORM started reformatting file filenam at the specified time.	Correct error and rerun.	DBFORM
CHAIN CONTROL ELEMENT MUST BE A GROUP OF TWO.	Self-explanatory.	Correct error and rerun.	DATADEF
CHANGE IN FILE TYPE OR CHAIN TYPE - filenam.	File filenam is defined as a different file type in the new EDT or it has changed from simple-chained to complex-chained or vice versa.	Correct error and rerun.	DBFORM
CONTAINS CLAUSE MUST PRECEDE KEY DEFINITION.	CONTAINS clause must appear before any of the following clauses. - PRIMARY-KEY - SECONDARY-KEY - RANDOM-KEY	Correct error and rerun.	DATADEF
CONTROL CARD DB NOT SAME AS LOAD FILE.	The data base name specified on the control statement does not match that encountered on the reformat load file.	Correct error and retry.	DBFORM
DATA BASE NAME MAY NOT START WITH *D* OR *T*.	Self-explanatory.	Correct error and rerun.	DATADEF
DATA BASE NAME MUST BE 2 CHARACTERS.	Self-explanatory.	Correct error and rerun.	DATADEF
DATA BASE NAME MUST START WITH A CHARACTER BETWEEN *A* AND *4*.	Only characters from A through Z and 0 through 4 are valid. SY is reserved for system usage.	Correct error and rerun.	DATADEF
DATA-FILE MUST BE OF TYPE *INDEXED*.	DATA-FILE must not be defined with RANDOM-KEY clause or REPEATED clause. However, PRIMARY-KEY clause must appear.	Correct error and rerun.	DATADEF
DATA MANAGER CEASED RESPONDING TO REQUESTS.	This message indicates software failure and should not occur.	Inform site analyst.	BDMI

MESSAGE	SIGNIFICANCE	ACTION	ROUTINE
DATE DIRECTIVE MISSING OR INCORRECT FOR LOAD.	The date on the input directive must be the same as the creation date on the load file.	Correct error and rerun.	DBFORM
DATE DIRECTIVE NEEDED FOR LOAD RECOVERY.	The date the load file was written must be specified on the date directive for a recovery of a load operation.	Correct error and rerun.	DBFORM
DATE DIRECTIVE PRESENT WITHOUT LOAD DIRECTIVE.	The DAT directive is valid only on a reformat load.	Remove the directive and retry.	DBFORM
DBFORM COMPLETE.	Informative message indicating the successful completion of DBFORM.	None.	DBFORM
DBFORM ERRORS.	DBFORM did not complete successfully because of fatal errors.	Examine the dayfile for error messages. Correct errors and retry.	DBFORM
DBFORM FATAL ERROR - ABORT.	DBFORM unsuccessfully attempted to attach a file or encountered an error as indicated in the dayfile.	Examine the dayfile. Correct error and retry.	DBFORM
DBFORM TABLE ERROR.	DBFORM cannot recognize a function in a table entry, indicating a system problem.	Inform site analyst.	DBFORM
DEF DIRECTIVE WITH NO NEW EDT.	A DEF directive specified a file which was not contained in the new EDT. Either the new EDT specified by the N parameter or the DEF directive was in error.	Correct error and rerun.	DBFORM
DESCRIPTION MUST APPLY TO A SIMPLE ELEMENT.	One or more attributes do not apply to key elements or group elements.	Correct error and rerun.	DATADEF
DIRECTIVE ERRORS.	Dayfile message indicating that one or more input directives were in error. Fatal error.	Examine output file to determine reason for error.	MODIFY OPLEDIT LIBTASK MODVAL PROFILE SYSEDIT
DIRECTIVE OTHER THAN DEFINE FOR-filenam.	Only the define directive is allowed for a file being added to the data base.	Correct error and rerun.	DBFORM
DUAL AND TRACE FLAGS FOR FILE filenam.	It is illegal to dual-record and trace the same file.	Correct error and reinitialize executive (TAF) or rerun job (BDMI).	TAF BDMI
DUAL RECORDED FILE MAY NOT BE TRACED.	Self-explanatory.	Correct error and rerun.	DATADEF
DUAL RECORDED FILE filenam NOT ATTACHED.	The user has neglected to attach file filenam.	Batch data manager users must attach all data files.	TAF BDMI

MESSAGE	SIGNIFICANCE	ACTION	ROUTINE
DUPLICATE CONTROL ELEMENT IN CHAIN FILE.	Only one group element may be used as a control element for a complex-chained file.	Correct error and rerun.	DATADEF
DUPLICATE DATA BASE NAME.	On a creation run the data base name specified in the DB parameter on the DBFORM statement was found in a DMS statement of the form DMS(TAF,ON,...) in file TCF.	Rename the new data base or remove the name from the DMS statement. Rerun.	DATADEF DBFORM
DUPLICATE ELEMENT NAME.	Within the same record, no two elements may have the same name.	Correct error and rerun.	DATADEF
DUPLICATE FILE NAME.	File names in a data base must be unique.	Correct error and rerun.	DATADEF
DUPLICATE FILE TYPE DESCRIPTION.	User tried to define file with attributes indicating two different file types.	Correct error and rerun.	DATADEF
DUPLICATE PRIMARY KEY DEFINITION.	Duplicate primary key defined in direct access file or indexed file.	Correct error and rerun.	DATADEF
DUPLICATE RANDOM KEY DEFINITION.	Self-explanatory.	Correct error and rerun.	DATADEF
DUPLICATE RECORD NAME.	All records in a data base must have unique names.	Correct error and rerun.	DATADEF
EDT FILE BAD.	Either the old EDT or the new EDT file is empty, or the data base name in either the old or new EDT does not agree with that specified by the DB parameter.	Correct error and rerun.	DBFORM
EDT NOT FOUND.	DBFORM cannot find the file containing the EDT for the data base specified on the DBFORM control statement.	Correct error and rerun.	DBFORM
EDT NOT PERMANENT FILE.	The new or old EDT files specified by the N and DB parameters on the control statement are not direct access permanent files.	Copy EDTs to direct access permanent files and retry.	DBFORM
ELEMENT FROM THIS FILE MUST MATCH LENGTH AND CHARACTER TYPE OF *PRIMARY-KEY* OF ABOVE FILE.	Elements in indexed or secondary file relationship definitions must match the corresponding elements in the appropriate files in both length and type of element.	Correct error and rerun.	DATADEF
ELEMENT LENGTH MUST BE .LE. 4080 BITS.	Self-explanatory.	Correct error and rerun.	DATADEF
ELEMENT MUST START WITH A CHARACTER BETWEEN *A* AND *4*.	Self-explanatory.	Correct error and rerun.	DATADEF
ELEMENT NAME MUST BE .LE. 3 CHARACTERS.	Self-explanatory.	Correct error and rerun.	DATADEF

MESSAGE	SIGNIFICANCE	ACTION	ROUTINE
EOF ENCOUNTERED IN CHAIN FILE - filenam.	No data was found on the initial read of file filenam.	The user should verify the existence of active records in the file.	DBFORM
EOF ENCOUNTERED IN INDEX FILE - filenam.	An EOF was encountered unexpectedly in the index file filenam.	The file should be checked to see if a system error occurred. Inform site analyst.	DBFORM
ERROR IN CONTROL CARD CALL.	Format of LGOB statement is incorrect.	Correct error and rerun.	BDMI
ERROR IN DBFORM ARGUMENTS.	Control statement error. Refer to DBFORM control statement.	Correct error and rerun.	DBFORM
FATAL DISK ERROR.	A fatal disk error was encountered.	Inform site analyst.	BDMI
FATAL ERROR STATUS RETURNED BY DATA MANAGER.	Data manager has found an error such that the user program cannot be notified. The user program is aborted.	Correct error and rerun.	BDMI
FILE ALREADY DEFINED - filenam.	DBFORM tried to define a file that was already defined.	Correct error and rerun.	DBFORM
FILE ALREADY PURGED - filenam.	DBFORM tried to purge a file that was previously purged.	Correct error and rerun.	DBFORM
FILE ALTERED - filenam.	Informative message indicating that the file has been reformatted and replaced.	None.	DBFORM
FILE CANNOT BE SHORTENED AS DIRECTED - filenam.	Active records exist at the end of the file filenam and cannot be removed.	Correct error and rerun.	DBFORM
FILE DEFINED - filenam.	Self-explanatory.	None.	DBFORM
FILE edt EMPTY.	Element descriptor table file edt is empty.	Correct error, reinitialize executive, and rerun.	TAF BDMI
FILE IN DEF DIRECTIVE NOT NEW FILE - filenam.	User has entered a DEF directive for a file which already exists.	Correct error and rerun.	DBFORM
FILE IN INPUT DIRECTIVE NOT IN EDT - filenam.	Input directives specify a file not described in the EDT.	Correct error and rerun.	DBFORM
FILE LENGTHENED - filenam.	Self-explanatory.	None.	DBFORM
FILE NAME MUST BE 4 CHARACTERS.	Self-explanatory.	Correct error and rerun.	DATADEF

MESSAGE	SIGNIFICANCE	ACTION	ROUTINE
FILE NAME MUST START WITH A CHARACTER BETWEEN *A* AND *4*.	Self-explanatory.	Correct error and rerun.	DATADEF
FILE NOT EMPTY - filenam.	Data base files must be empty for a data base creation run.	Correct error and rerun.	DBFORM
FILE NOT PERMANENT - filenam.	A local or common file exists at the control point with the same name as one to be defined by DBFORM.	Correct error and rerun.	DBFORM
FILE NOT PERMANENT-NO ACTION - filenam.	During a reformat load an attempt was made to lengthen or shorten a file that was not a direct access pernament file. The reformat load continues.	If a file was lost, data base recovery action is needed.	DBFORM
FILE PREALLOCATED - filenam.	Self-explanatory.	None.	DBFORM
FILE PURGED - filenam.	Self-explanatory.	None.	DBFORM
FILE SHORTENED - filenam.	Self-explanatory.	None.	DBFORM
FILE SPECIFIED AS EDT FILE IS NOT EDT FILE TYPE.	The file specified as the EDT is not an EDT.	Correct error and rerun, or inform site analyst.	TAF BDMI
FILE edt TOO LARGE.	Element descriptor table is too large to be read into memory.	Correct error and reinitialize executive (TAF), or rerun (BDMI).	TAF BDMI
FILE UPDATED - filenam.	Self-explanatory.	None.	DBFORM
FINISHED REFORMAT - filenam hh.mm.ss.	DBFORM finished reformatting file filenam at the specified time.	None.	DBFORM
FIRST PARAMETER NOT DATA BASE NAME.	The first parameter following the period on the LGOB card must be the data base name of the data base that the batch job is expected to access.	Correct error and rerun.	BDMI
FORMAT ERROR ON DATE DIRECTIVE-yy/mm/dd.	Refer to Input Directives in section 4.	Correct error and rerun.	DBFORM
GROUP ELEMENT HAS NO FILE ASSOCIATION.	Record containing group element must be used in some file.	Correct error and rerun.	DATADEF
ID ERROR, NOT DBFORM LOAD FILE.	The tape assigned was not written by DBFORM.	Assign proper load file.	DBFORM
ILLEGAL CHARACTER IN NUMBER OF RECORDS.	Record number must be numeric on SRF directive.	Correct error and rerun.	DBFORM
ILLEGAL COMBINATION OF ACTIONS ON - filenam.	User may not define and shorten, define and remove inactive records, define and write by chain, or shorten and remove inactive records on one file in one run.	Correct error and rerun.	DBFORM

MESSAGE	SIGNIFICANCE	ACTION	ROUTINE
ILLEGAL DELIMITER OR SEPARATOR ON DIRECTIVE.	User has forgotten a slash or comma or substituted an illegal character.	Correct error and rerun.	DBFORM
ILLEGAL DEVICE.	For legal devices, refer to Input Directives in section 4.	Correct error and rerun.	DBFORM
ILLEGAL EQUIPMENT.	For equipment number definition refer to Input Directives in section 4.	Correct error and rerun.	DBFORM
ILLEGAL FILENAME.	The first character of the file name specified was not alphabetic, or subsequent characters were not alphanumeric.	Correct error and rerun.	DBFORM
ILLEGAL NUMERIC CONSTANT.	Numeric constant expected is out of bounds or unrecognizable.	Correct error and rerun.	DATADEF
ILLEGAL RUN TIME REQUEST RECEIVED.	An illegal function code was specified on CTI or SCT request.	Correct program and rerun.	BDMI
INACTIVE RECORD SPACE REMOVED FROM - filenam.	Self-explanatory.	Correct error and rerun.	DBFORM
INCOMPLETE STATEMENT.	More information needed in statement.	Correct error and rerun.	DATADEF
INCORRECT LENGTH IN TABL -NO ACTION - filenam.	The number of records in file filenam after a lengthening or shortening in its table entry on the load file is invalid. System error.	Inform site analyst.	DBFORM
INDEX ENTRY COUNT MUST BE .LE. 16777215.	Self-explanatory.	Correct error and rerun.	DATADEF
INDEX FILE MAY NOT HAVE KEYS.	Self-explanatory.	Correct error and rerun.	DATADEF
INSUFFICIENT FL FOR DBFORM.	Self-explanatory.	Increase FL and retry.	DBFORM
KEY ALREADY USED - keyvalue.	System error. A duplicate key value has been found while updating an index file.	Inform site analyst.	DBFORM
KEY DOES NOT WORK WITH ACCESS METHOD - keyvalue.	The result of using key value to obtain a data file record results in an invalid PRU number.	Correct error and rerun.	DBFORM
KEY LENGTH MUST BE .GE. 1 AND .LE. 120 BITS.	Self-explanatory.	Correct error and rerun.	DATADEF
KEY MAY NOT BE A GROUP ELEMENT.	Primary keys, secondary keys, and random keys must not be group elements.	Correct error and rerun.	DATADEF

MESSAGE	SIGNIFICANCE	ACTION	ROUTINE
KEYWORD UNKNOWN OR INVALID IN THIS SECTION.	Self-explanatory.	Correct error and rerun.	DATADEF
LOAD FILE BAD - ABORT.	Reformat load has encountered a table entry with no function code on the load file. System error.	Inform site analyst.	DBFORM
LOAD PRESENT WITHOUT DATE DIRECTIVE - ABORT.	Date directive must be present for load unless loading in the same run as dumping.	Correct error and rerun.	DBFORM
LOCATION-WORD DECLARATION MUST PRECEDE *START-BIT* DESIGNATION.	Self-explanatory.	Correct error and rerun.	DATADEF
LOCATION WORD MUST BE .GE. 1 AND .LE. 1024.	Self-explanatory.	Correct error and rerun.	DATADEF
NEW EDT NOT FOUND.	DBFORM cannot find the file equivalenced to N on the DBFORM control statement.	Correct error and rerun.	DBFORM
NO ACCESS METHOD DEFINED YET.	ACCESS-METHOD clause must precede BIAS-VALUE clause.	Correct error and rerun.	DATADEF
NO DATA BASE NAME DEFINED.	Self-explanatory.	Correct error and rerun.	DATADEF
NO *DATA-FILE* DEFINED YET.	DATA-FILE clause must precede PRIMARY-INDEX clause or SECONDARY-KEY clause.	Correct error and rerun.	DATADEF
NO EDT FOUND ON LOAD FILE.	The file specified for the load was not written by DBFORM.	Correct error and rerun.	DBFORM
NO ELEMENT DEFINED YET.	Element name must be defined first in element definition.	Correct error and rerun.	DATADEF
NO ELEMENTS DEFINED.	Self-explanatory.	Correct error and rerun.	DATADEF
NO FILE DEFINED YET.	Self-explanatory.	Correct error and rerun.	DATADEF
NO FILES DEFINED.	Self-explanatory.	Correct error and rerun.	DATADEF
NO INDEX FILE SPECIFIED FOR THIS DATA FILE.	Self-explanatory.	Correct error and rerun.	DATADEF
NO PARAMETERS ON CONTROL CARD.	The LGOB control statement contains no parameters.	Correct error and rerun.	BDMI
NO PRIMARY KEY, EDT ERROR - filenam.	A data file exists in the data base without an element defined as its key.	Rerun DATADEF to get a new EDT.	DBFORM

MESSAGE	SIGNIFICANCE	ACTION	ROUTINE
NO *PRIMARY-KEY* FOR ABOVE *DATA-FILE*.	Self-explanatory.	Correct error and rerun.	DATADEF
NO RECORD DEFINED YET.	A record must be defined before an element in it may be defined.	Correct error and rerun.	DATADEF
NO REFORMAT - NO RECOVERY.	The R option was not selected so no recovery is possible.	Correct error and rerun.	DBFORM
NO TAPE LABEL.	User is trying to load from a dump tape with no label.	Mount DBFORM dump tape.	DBFORM
NOT A *SECONDARY-KEY* FOR ABOVE *DATA-FILE*.	Element indicated as secondary key is either not defined as a secondary key in the record definition section or is misspelled.	Correct error and rerun.	DATADEF
NUMBER OF RECORDS ON SRF LARGER THAN FILE - filenam.	File filenam is not as large as the number of records specified in SRF directive.	Correct error and rerun.	DBFORM
NUMBER OF RECORDS OVERFLOWS FIELD.	The number of records specified on an SRF directive was greater in length than eight display digits or eight octal digits.	Correct error and rerun.	DBFORM
OLD EDT REPLACED.	The new EDT is stored in place of the old EDT.	None.	DBFORM
P.F. ERROR - ABORT.	Uncommon error received from PFM. A possible example is a PFM error on an ATTACH other than file busy or file not found.	Correct error and rerun.	DBFORM
PREALLOCATING filenam.	Self-explanatory.	None.	DBFORM
PREVIOUS *RANDOM-KEY* OR *PRIMARY-KEY* CLAUSE REQUIRED FOR *SECONDARY-KEY*.	Self-explanatory.	Correct error and rerun.	DATADEF
PRIMARY-INDEX FILE MUST BE OF TYPE *HASHED-ACCESS-INDEX* OR *DIRECT-ACCESS-INDEX*.	Self-explanatory.	Correct error and rerun.	DATADEF
PRIMARY KEY REPETITION MUST BE .LE. 16777215.	Self-explanatory.	Correct error and rerun.	DATADEF
READ SECURITY MUST BE .LE. 7.	Self-explanatory.	Correct error and rerun.	DATADEF
RECORD NAME MUST BE .LE. 7 CHARACTERS.	Self-explanatory.	Correct error and rerun.	DATADEF
RECORD NAME MUST START WITH A CHARACTER BETWEEN *A* AND *4*.	Self-explanatory.	Correct error and rerun.	DATADEF

MESSAGE	SIGNIFICANCE	ACTION	ROUTINE
RECORDS TOO LONG TO REFORMAT - filenam.	The length of a logical record in file filenam is longer than 1777B words.	Correct error and rerun.	DBFORM
RECOVERY DATE NOT SAME AS CHECKPOINT.	The date on the recovery file does not match the current date.	Correct error and rerun.	DBFORM
RECOVERY DB NOT SAME AS CHECKPOINT.	The data base name on the recovery file is not the same as on the load file.	Correct error and rerun.	DBFORM
RECOVERY IMPOSSIBLE.	Recovery information is either not available or is garbled.	Correct error and rerun.	DBFORM
RECOVERY IMPOSSIBLE.	Recovery information is either not available or is garbled.	Correct error and rerun.	DBFORM
REEL OUT OF SEQUENCE.	The wrong reel was assigned for loading.	Assign proper reel.	DBFORM
REPEATED CLAUSE REQUIRES PREVIOUS *PRIMARY-KEY*.	Self-explanatory.	Correct error and rerun.	DATADEF
RESERVED DATA BASE NAME.	Data base name must not be SY.	Correct error and rerun.	DATADEF
RESERVED FILE NAME.	A reserved file name was incorrectly used.	Choose a nonreserved file name.	OPLEDIT EDIT DATADEF IAFEX TELEX
RIR ON RANDOM OR PRE-ALLOCATED FILE - filenam.	Inactive record space cannot be removed from a random or preallocated file except at the end.	Correct error and rerun.	DBFORM
SECTION DUPLICATED OR OUT OF ORDER.	Sections must be in the following order. IDENTIFICATION SECTION RECORD-DESCRIPTION SECTION FILE-DESCRIPTION SECTION FILE-RELATIONSHIP SECTION	Correct error and rerun.	DATADEF
SRF ON A NON-RANDOM FILE - filenam.	This directive is for random files only.	Correct error and rerun.	DBFORM
START BIT MUST .LE. 59.	Self-explanatory.	Correct error and rerun.	DATADEF
STORAGE MODE *D* ELEMENT LENGTH MUST BE 120 BITS.	Self-explanatory.	Correct error and rerun.	DATADEF
STORAGE MODE *D* ELEMENT LENGTH MUST START IN BIT 59.	Self-explanatory.	Correct error and rerun.	DATADEF

MESSAGE	SIGNIFICANCE	ACTION	ROUTINE
STORAGE MODE *F* ELEMENT LENGTH MUST BE 60 BITS.	Self-explanatory.	Correct error and rerun.	DATADEF
STORAGE MODE *F* ELEMENT MUST START IN BIT 59.	Self-explanatory.	Correct error and rerun.	DATADEF
STORAGE MODE *I* ELEMENT LENGTH MUST BE .GE. 0 AND .LE. 60 BITS.	Self-explanatory.	Correct error and rerun.	DATADEF
STORAGE MODE *I* ELEMENT MUST FIT ENTIRELY IN ONE WORD.	Self-explanatory.	Correct error and rerun.	DATADEF
STORAGE MODE *L* ELEMENT LENGTH MUST BE .GE. 0 AND .LE. 60 BITS.	Self-explanatory.	Correct error and rerun.	DATADEF
STORAGE MODE *R* ELEMENT LENGTH MUST BE 60 BITS.	Self-explanatory.	Correct error and rerun.	DATADEF
STORAGE MODE *R* ELEMENT MUST START IN BIT 59.	Self-explanatory.	Correct error and rerun.	DATADEF
STORAGE MODE *U* ELEMENT LENGTH MUST BE .GE. 0 AND .LE. 60 BITS.	Self-explanatory.	Correct error and rerun.	DATADEF
STORAGE MODE *U* ELEMENT MUST FIT ENTIRELY IN ONE WORD.	Self-explanatory.	Correct error and rerun.	DATADEF
STORAGE MODE *X* ELEMENT LENGTH MUST BE .GE. 0 AND .LE. 4080 BITS.	Self-explanatory.	Correct error and rerun.	DATADEF
STORAGE MODE *X* ELEMENT MUST START ON A CHARACTER BOUNDARY.	Self-explanatory.	Correct error and rerun.	DATADEF
STORAGE MODE *9* ELEMENT LENGTH MUST BE .GE. 0 AND .LE. 4080 BITS.	Self-explanatory.	Correct error and rerun.	DATADEF
STORAGE MODE *9* ELEMENT MUST START ON A CHARACTER BOUNDARY.	Self-explanatory.	Correct error and rerun.	DATADEF
SYNTAX ERROR - DEVICE NOT SPECIFIED.	Device type must be specified on a DEF directive.	Correct error and rerun.	DBFORM
SYNTAX ERROR - ILLEGAL TERMINATOR ON DIRECTIVE.	Refer to Input Directives in section 4.	Correct error and rerun.	DBFORM
SYNTAX ERROR - NO SLASH IN DATE.	Format on DAT directive incorrect. Refer to Input Directives in section 4.	Correct error and rerun.	DBFORM
SYNTAX ERROR - NOT ENOUGH INFORMATION ON DIRECTIVE.	One or more parameters are missing. Refer to Input Directives in section 4.	Correct error and rerun.	DBFORM

MESSAGE	SIGNIFICANCE	ACTION	ROUTINE
SYNTAX ERROR - NUMBER OF RECORDS NOT SPECIFIED.	Number of records must be specified on a directive to shorten a random file.	Correct error and rerun.	DBFORM
TABLE ERROR-FILE NOT IN EDT NO ACTION.	Possible software malfunction.	Inform site analyst.	DBFORM
TIME DELAY FOR LOAD ILLEGAL.	The load file is too old to load from.	Increase the assembly constant DLAG if desired.	DBFORM
TRACED FILE MAY NOT BE DUAL RECORDED.	Self-explanatory.	Correct error and rerun.	DATADEF
UNKNOWN ATTACH MODE.	Self-explanatory.	Correct error and rerun.	DATADEF
UNKNOWN DIRECTIVE.	Input directive is not recognized by DBFORM.	Correct error and rerun.	DBFORM
UNKNOWN ELEMENT NAME.	Element is not defined or is misspelled.	Correct error and rerun.	DATADEF
UNKNOWN FILE NAME.	File is not defined or is misspelled.	Correct error and rerun.	DATADEF
UNKNOWN LENGTH UNIT.	Length unit must be one of the following. BIT BITS CHARACTER CHARACTERS WORD WORDS	Correct error and rerun.	DATADEF
UNKNOWN RECORD NAME.	Record is not defined or is misspelled.	Correct error and rerun.	DATADEF
UNKNOWN STORAGE MODE.	Storage mode must be one of the following. D, F, I, L, R, U, X, or 9.	Correct error and rerun.	DATADEF
UNRECOGNIZABLE CIO ERROR CODE.	CIO returned unrecognizable status code to the data manager.	Correct error and rerun.	BDMI TAF
UPDATE SECURITY MUST BE .LE. 7.	Self-explanatory.	Correct error and rerun.	DATADEF
USER DEFINED A FILE AND HAS DIRECTIVE TOO - filenam.	No directive is allowed for a file created in the current run other than DEF.	Correct error and rerun.	DBFORM
WBC DIRECTIVE FOR INDEX FILE - filenam.	Only data files can be written by chain.	Correct error and rerun.	DBFORM
WBC FOR ILLEGAL FILE TYPE-filenam.	A WBC directive was specified for a complex-chained random or a complex-chained direct-key-access file.	Correct error and rerun.	DBFORM

MESSAGE	SIGNIFICANCE	ACTION	ROUTINE
WRITE PARITY ERROR UNRECOVERED.	An unrecovered parity error occurred while writing on the dump file.	Rerun DBFORM with a new tape.	DBFORM
filenam ADDRESS ERROR.	Error detected by CIO.	Inform site analyst.	BDMI TAF
filenam DEVICE NOT READY.	Error detected by CIO.	Inform site analyst.	BDMI TAF
filenam DEVICE RESERVED.	Error detected by CIO.	Inform site analyst.	BDMI TAF
filenam DEVICE STATUS ERROR.	Error detected by CIO.	Inform site analyst.	BDMI TAF
filenam FUNCTION REJECT.	Error detected by CIO.	Inform site analyst.	BDMI TAF
filenam PARITY ERROR.	Error det d by CIO.	Inform site analyst.	BDMI TAF
filenam TRACK LIMIT.	Error detected by CIO.	Inform site analyst.	BDMI TAF

The sample deck structures in this appendix demonstrate the use of various parameters which are available. The default input and output files can be used to reduce the number of control language statements needed to bring about a compile, load, and LIBTASK sequence, because the default binary output files are the binary input files for subsequent processors.

For example:

```
COBOL5,ANSI=77LEFT,UC1.
LOAD(LGO)
NOGO(LGOB)
LIBTASK,P=RLTASKL,TT.
```

It is assumed that each of the following decks is preceded by job and USER statements. Some sites may require a CHARGE statement.

BATCH TAF DATA MANAGER COMPASS DECK STRUCTURE

The control statements for a batch COMPASS program that accesses data base DB are illustrated in figure E-1. (Trace information is written on file TRACE.)

```
COPYBR(INPUT,BATCH)
REWIND(BATCH)
ATTACH(OPL/UN=usernum)
MODIFY(Z)/*CREATE,BATCH/*EDIT,BATCH
ATTACH(DB/M=R)
ATTACH(DBFIL1/M=W)
ATTACH(DBFIL2/M=W)
ATTACH(DBIND1/M=W)
ATTACH(DBIND2/M=W)
COMPASS(I)
LDSET(LIB=BDMLIB)
LOAD(LGO)
NOGO(LGOB)
LGOB.DB,*.
REWIND,TRACE.
COPYSBF,TRACE,OUTPUT.
7/8/9
BATCH
        IDENT     BATCH
        ENTRY     BATCH
*CALL   COMKMAC
        USE       /COM/COMBLK
        BSS       69
        USE       *
        .
        .<------COMPASS task appears here
        .
6/7/8/9
```

Figure E-1. Batch COMPASS Deck Structure

NOTE

The system OPL must be attached for COMKMAC to be available in the COMPASS program. Consult site personnel for the appropriate control statement.

Files associated with the data base must be attached by the user when running a batch job.

BATCH TAF DATA MANAGER FORTRAN DECK STRUCTURE

The control statements for a batch FORTRAN program that accesses data base DB are illustrated in figure E-2. (Trace information is written on file TRACE.)

```
ATTACH(DB/M=R)
ATTACH(DBFIL1/M=RM)
ATTACH(DBFIL2/M=W)
ATTACH(DBIND1/M=RM)
ATTACH(DBIND2/M=W)
FTN.
LDSET(LIB=BDMLIB)
LOAD(LGO)
NOGO(LGOB)
LGOB.DB,*.
REWIND(TRACE)
COPYSBF(TRACE)
7/8/9
    .
    .<------FORTRAN task appears here
    .
6/7/8/9
```

Figure E-2. Batch FORTRAN Deck Structure

NOTE

Files associated with the data base must be attached by the user when running a batch job.

BATCH TAF DATA MANAGER COBOL DECK STRUCTURE

The control statements for a batch COBOL program that accesses data base DB are illustrated in figure C-3. (Trace information is written on file TRACE.) The control statements for a batch COBOL program that uses the paragraph trace facility are illustrated in figure C-4.

```
ATTACH(DB/M=R)
ATTACH(DBFIL1/M=RM)
ATTACH(DBFIL2/M=W)
ATTACH(DBIND1/M=RM)
ATTACH(DBIND2/M=W)
COBOL5,ANSI=77LEFT,UC1.
LDSET(LIB=BDMLIB)
LOAD(LGO)
NOGO(LGOB)
LGOB.DB,*.
REWIND,TRACE.
COPYSBF,TRACE,OUTPUT.
7/8/9
    .
    . <------ COBOL task appears here
    .
6/7/8/9
```

Figure E-3. Batch COBOL Deck Structure

NOTE

Files associated with the data base must be attached by the user when running a batch job.

```
COBOL5,ANSI=77LEFT,DB=TR,UC1.
LDSET(LIB=TRANLIB)
LOAD(LGO)
NOGO(LGOB)
LIBTASK(P=RLTASKL,TT,I=0)
7/8/9
    .
    .<---COBOL task appears here
    .
6/7/8/9
```

Figure E-4. Batch COBOL Deck Using Paragraph Trace

NOTE

Under COBOL, the PROGRAM-ID name is used for the task name. Only alphanumeric names can be used as task names.

INDEX

ACCESS-METHOD, BIAS-VALUE 3-7,8
Add records 6-5
ADDR 6-5
ATTACH-MODE 3-7,8

Batch access to data manager 6-17
BDMI 6-17
BDMLIB 6-19
BLKGET 6-5
BLKPUT 6-5
Block get 6-5
Block put 6-5
BST status A-1

CEASE 5-3
CHAIN 3-10
Chain area 6-1
Chain descriptor 6-1
CHAIN FILE-NAME 3-9
Chaining 2-1,2
CLEAR 6-5
COBOL data types considerations 6-4
COMMENT statement 3-2
Complex chaining 2-2
CONTAINS 3-7,8
Control program 5-1
Count 6-1
CRAT 5-6
Create data base 4-1,2
Create option 4-2
CTI requests 6-17

D storage mode 3-6
D dump file for reformat option 4-2,4
DAT directive 4-4
Data area 6-2
Data base formation 4-1
Data base map 3-2,4
Data base name 6-1
DATADEF utility 3-1
DATA-FILE 3-9
DATAMAP utility 3-2,4
DBA 5-1; 6-17
DBFORM 4-1
DEF directive 4-4
Diagnostics D-1
DIRECT-ACCESS-INDEX 3-7,8
Direct-key-access indexed file 2-1,5
DMSTAT 5-3; 6-6; A-1
Dual recording 5-5
DUAL-RECORD 3-10
Dump option 4-4

EDT address 6-1
EDT buffer size 6-1
EDT file 3-1; 4-1
EJECT statement 3-2
ELEMENT 3-5,6,9
Element area 6-2
Element conversion 5-4; B-1
Element descriptor 6-2
Element processor 5-4
Entry count 3-8
ENTRIES 3-7,8
Error messages D-1
ERPF 5-6

F storage mode 3-6
File access processor 5-2
File description section 3-7
File name 6-2
File relationship section 3-9
File types 2-4
FILE-NAME 3-7
Function processors 5-2

Get elements 6-8
Get elements backward 6-6
Get elements backward locked 6-6
Get elements locked 6-8
Get next record 6-7
Get next record locked 6-8
Get record 6-7
Get record backward 6-8
Get record backward locked 6-8
Get record locked 6-7
GETB 6-6
GETBL 6-6
GETL 6-8
GETN 6-7
GETNL 6-7
GETNR 6-7
GETNRL 6-8
GETR 6-7
GETRB 6-8
GETRL 6-7
GETT 6-8
GROUP ELEMENT 3-7

HASHED-ACCESS-INDEX 3-7,8
Hashed-access-indexed file 2-1,7

I input directive file 4-2,4
I storage mode 3-6
Identification section 3-5
IMPLIED-LENGTH 3-7
IMPLIED-READ-SECURITY 3-7
IMPLIED-STORAGE-MODE 3-7
IMPLIED-UPDATE-SECURITY 3-7
Input directive option 4-4
Input to DATADEF 3-4

Key area 6-2
Key buffer area 6-2
Key conversion access methods C-1
Key descriptor 6-2

L load file for reformat option 4-2,4
L storage mode 3-6
Length 6-2
LENGTH 3-6
LGOB 6-17
Load option 4-4
LOCATION -WORD 3-7
Lock a file 6-8
LOCKF 5-3; 6-9
LO option 4-2,4

Messages D-1

N new EDT file option 4-2,4
NREC 5-1,2,3; 6-5,6
Number 6-2
Number of records 6-2

Off tracing 6-9
OFFTRCE 6-9
On tracing 6-9
ONTRACE 6-9
Open new record; refer to ADDR

Pool files 5-6
PREFIX statement 3-2
PRIMARY-INDEX 3-9
PRIMARY-KEY 3-7,8
PRIME-KEY 3-9
Purge option 4-2,3
Purge data base 4-1,3
Purge records 6-9
PURGER 6-9
PUT 6-9
Put elements 6-9
Put elements forced 6-10
Put elements inserted 6-10
Put elements inserted forced 6-10
Put records 6-10
Put records forced 6-11
Put records inserted 6-11
Put records inserted forced 6-11
PUTF 6-10
PUTI 6-10
PUTIF 6-10
PUTR 6-10
PUTRF 6-11
PUTRI 6-11
PUTRIF 6-11

R recovery parameters 4-2,4
R storage mode 3-6
Random file 2-1,5
RANDOM-KEY 3-7,8
READ-SECURITY 3-7
READEDT 5-3,4; 6-11
Recall 5-5; 6-2
RECHAIN 5-3; 6-12
Record accessing techniques 2-3
Record-description section 3-5
Record status entry 5-3; 6-2
RECORD-NAME 3-6
Recovery option 4-4
Reformat data base 4-1,3
Reformat option 4-2,3
Release buffer space 6-12
RELES 5-3; 6-12
RELESAL 5-3; 6-12
REPEATED 3-7,8
REPOS 6-11
Reposition file 6-11
Return flag 6-2
Rewind; refer to REPOS
RIR directive 4-4
RSE 5-3; 6-2
RSE fuffer 6-2

SCT requests 6-17
Secondary keys 2-1,2
SECONDARY-KEY 3-7,8,9
Security 1-1
Sequential file 2-1,4
Simple chaining 2-1
SPACE statement 3-2
SRF directive 4-4
START-BIT 3-7
STATCHK 6-13
Status bit 6-3
Status bit buffer 6-3
Status checking 6-13
Status word 6-3
STORAGE-MODE 3-5

TITLE statement 3-2
TRACE 3-10
Trace files 5-6
Trace option, batch data manager 6-17
TRANLIB 6-19
Trial reformat 4-2,4
Turn off tracing 6-9
Turn on tracing 6-9
Type 6-3

U storage mode 3-6
UNLOCK 5-3
Unlock a record 6-13
UNLOK 5-3; 6-13
UNLOKAL 6-13
UNLOKF 5-3; 6-13
UPDATE-SECURITY 3-7

WBC directive 4-4

X storage mode 3-6

9 storage mode 3-6

COMMENT SHEET

MANUAL TITLE: CDC Network Products Transaction Facility Version 1
Data Manager Reference Manual

PUBLICATION NO.: 60455350 REVISION: C

NAME:_____

COMPANY:_____

STREET ADDRESS:_____

CITY:_____ STATE:_____ ZIP CODE:_____

This form is not intended to be used as an order blank. Control Data Corporation welcomes your evaluation of this manual. Please indicate any errors, suggested additions or deletions, or general comments below (please include page number references).

☐ Please Reply ☐ No Reply Necessary

CUT ALONG LINE

AA3419 REV 1/80 PRINTED IN U.S.A.

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

FOLD ON DOTTED LINES AND TAPE. POSTAL REGULATIONS PROHIBIT THE USE OF STAPLES.

OLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 8241 MINNEAPOLIS, MINN.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION

Publications and Graphics Division
ARH219
4201 North Lexington Avenue
Saint Paul, Minnesota 55112

CUT ALONG LINE

OLD FOLD

TITLE: CDC Network Products Transaction Facility Version 1
Data Manager Reference Manual

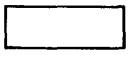
PUBLICATION NO.: 60455350
REVISION: C

External A/D
CONTROL DATA CORPORATION
Publications and Graphics Division
4201 North Lexington Avenue
St. Paul, Minnesota 55112

REASON FOR CHANGE:

DATE 12-12-80

Revised to update manual to NOS 1.4 at PSR level 531. This edition obsoletes all previous editions.



CORPORATE HEADQUARTERS, P.O. BOX 0, MINNEAPOLIS, MINN. 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

LITHO IN U.S.A.

CONTROL DATA CORPORATION